Titre: Title:	Placement et chaînage dynamique de fonctions réseau sur multiples centres de données
Auteur: Author:	Reuel Nathan Wandji Ngassam
Date:	2018
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Wandji Ngassam, R. N. (2018). Placement et chaînage dynamique de fonctions réseau sur multiples centres de données [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/3698/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/3698/
Directeurs de recherche: Advisors:	Brunilde Sanso
Programme: Program:	génie électrique

UNIVERSITÉ DE MONTRÉAL

PLACEMENT ET CHAÎNAGE DYNAMIQUE DE FONCTIONS RÉSEAU SUR MULTIPLES CENTRES DE DONNÉES

REUEL NATHAN WANDJI NGASSAM
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
NOVEMBRE 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PLACEMENT ET CHAÎNAGE DYNAMIQUE DE FONCTIONS RÉSEAU SUR MULTIPLES CENTRES DE DONNÉES

présenté par : WANDJI NGASSAM Reuel Nathan
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. CARDINAL Christian, Ph. D., président
Mme SANSO Brunilde, Ph. D., membre et directrice de recherche
M. GIRARD André, Ph. D., membre

DÉDICACE

À ma maman et à mon papa,
à ma famille de Montréal,
merci pour tout...

REMERCIEMENTS

Je tiens à remercier mes chers parents : ma mère Rosaline Yongoua et mon père Rigobert Ngassam pour leur soutien psychologique continuel.

Je remercie aussi ma famille à Montréal : Jules Yansa et sa femme Eliane Kepya pour leur accompagnement et leur soutien qui ont eu un impact incommensurable depuis mon arrivée et durant ma maîtrise. Merci de tout coeur. Je tiens à remercier tout membre de ma famille qui de près ou de loin a contribué à ma reussite.

Je remercie énormément ma directrice de recherche Brunilde Sansò pour son suivi, sa patience, son engagement, ses encouragements et ses conseils.

Je remercie tous les membres du jury pour l'intérêt porté à l'évaluation de mon projet.

Je remercie mes collègues de laboratoire : Mathieu, Filippo, Issoufou et Orestes pour leur aide et les échanges constructifs durant toute ma période de recherche.

Je remercie le personnel et les chercheurs du GERAD pour le soutien matériel et intellectuel que j'ai reçu.

RÉSUMÉ

La multiplicité et la diversité d'applications dans les réseaux de télécommunications ont causé leur évolution rapide et leur complexification. Pour pouvoir donc diminuer et leurs coûts d'installation et leur complexité, les opérateurs ont exploité la virtualisation des fonctions réseaux et leur chaînage. Ceci est possible grâce à la portabilité et à la flexibilité qu'apportent les fonctions virtuelles. Mais pour pouvoir utiliser ces chaînes de fonctions virtualisées, il faut les héberger dans des centres de données, et ce, le plus efficacement possible. Bien que l'optimisation du placement des fonctions virtuelles soit présent dans la littérature, nous avons apporté des contributions originales. Nous avons tenu compte du fait que les prix des hébergements dans les centres de données varient avec le temps, que les délais de transmission ne sont pas les mêmes tout le temps et que les demandes en chaînes changent aussi. De plus, il existe des relations entre les chaînes et les fonctions telles que la symétrie, l'affinité, la migration et l'incompatibilité qui ont été spécifiquement modélisés dans notre travail.

Nous avons proposé un nouveau modèle sous la forme d'un programme linéaire en nombre entiers qui minimise le coût de placement d'une chaîne dans un réseau de centre de données sur une journée en tenant compte de la qualité de service, des capacités des centres de données et des interdépendances entre les chaînes et les fonctions.

Pour résoudre les grandes instances du problème, nous avons développé des heuristiques gloutonne et tabou et nous avons étudié leur comportement en fonction du type de fonctions dans les centres de données.

ABSTRACT

The multiplicity and diversity of applications in telecommunication networks have caused their rapid evolution, and have increased network complexity. To reduce network installation costs and complexity, operators have exploited the virtualization of network functions and their chaining. This is possible because of the portability and flexibility that virtual functions bring. However, to be able to use these chains of virtualized functions, it is necessary to host them in data centers as efficiently as possible. Although the optimization of service function chains is present in the litterature, we have made some original contributions in this work. We take into account the fact that datacenter hosting prices vary over time, that transmission delays are not the same all the time and that chain throughput demands change as well. In addition, there are relationships between chains and functions such as symmetry, incompatibility, affinity and migration that have been specifically modeled in this work.

We propose an integer linear program that minimizes the cost of placing a service function chain in a data center network over a day, taking into account the quality of service, the capabilities of the datacenters and interdependencies between chains and functions.

To solve big instances of the problem, we develop greedy and tabu search heuristics and we study their behavior according to the type of functions in the datacenters.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AMPL	A Mathematical Programming Language
API	Application Programming Interface
AWS	Amazon Web Service
CAPEX	Capital Expendature
CPU	Computer Processing Unit
DPI	Deep Packet Inspector
ETSI	European Telecommunication Standards Institute
FP	Fonction Physique
FV	Fonction Virtuelle
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
ILP	Integer Linear Program
INLP	Integer Non Linear Program
ISG	Industry Specification Group
MANO	Management and Orchestrator
MILP	Mixed Integer Linear Program
MIQCP	MIexed Integer Quadratically Contrained Program
NDN	Named Deviced Networking
NFV	Network Function Virtualization
ONF	Open Networking Foundation
OPEX	Operational Expendature
PDP	Point de Présence
POP	Point of Presence
RAM	Random Access Memory
RFC	Request For Comments
SDN	Software Defined Networking
SFC	Service Function Chaining
SLA	Service Level Agreement
SLO	Service Level Objective
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VNE	Virtual Network Embeeding

CHAPITRE 1 INTRODUCTION

1.1 Introduction

La vulgarisation de l'Internet entamée dans les années 90 est aujourd'hui une réussite. Cette réussite s'accompagne de beaucoup de défis. En effet, l'abondance et la diversité des applications Internet rendent le réseau complexe et très chargé. La rédaction du Journal du Net (2018) a publié les statistiques d'utilisation d'Internet. Les données présentées montrent une augmentation exponentielle du nombre d'utilisateurs entre 1995 et 2017. Cette augmentation a bien des effets dont la virtualisation des fonctionnalités du réseau. Dans ce chapitre, nous présenterons le contexte et la problématique du chainage des fonctions virtuelles, puis nous présenterons les objectifs de notre recherche ainsi que les hypothèses que nous avons considéré et enfin nous conclurons.

1.2 Contexte

À l'origine d'Internet, le but des organisations était de pouvoir partager des informations pour la recherche. Ce but a évolué quand l'Internet s'est ouvert au grand public dans la deuxième moitié des années 1990. Pendant cette période, l'utilisation d'Internet se faisait le plus souvent à travers le web. Les utilisateurs se servaient d'un navigateur pour se connecter et accéder à des pages qui étaient pour la plupart statiques. Avec l'exploitation des bases de données et la création des langages de programmation dynamiques, les contenus dynamiques ont vu le jour. Ce qui a permis aux entreprises de fournir des meilleurs contenus et services, et entrainé une utilisation croissante d'Internet. Ceci a crée d'innombrables défis pour que le réseau Internet des opérateurs puisse soutenir la demande. Avec la demande grandissante, les organisations ont commencé à fournir des services plus complexes et risqués. Les opérateurs devaient donc non seulement répondre rapidement aux requêtes d'utilisateurs, mais aussi assurer leur intégrité, la sécurité des utilisateurs et la fiabilité du service.

Pour assurer l'intégrité, la fiabilité et la sécurité sur Internet, d'avantage d'équipements spécialisées ont été créés, les uns plus performant que les autres. Au fur et à mesure que le réseau s'agrandissait, ceux-ci coûtaient de plus en plus cher non seulement à l'achat, mais aussi pendant leur exploitation. L'utilisation d'un très grand nombre d'équipements différents en rend la configuration complexe, difficile et chère. En effet puisque les équipements sont souvent de fournisseurs différents, cela requiert beaucoup d'investissement en formations pour acquérir des connaissances pour les exploiter. De plus, une panne dans le réseau avait

un très gros impact sur celui-ci. Ceci s'ajoutait au fait que la découverte de la panne peut être très compliquée ce qui peut entraîner de longues interruptions de services.

Les avancées dans le domaines de l'infonuagique ont poussé l'amélioration de la virtualisation. En effet, pour faire face à l'épuisement de ressources et permettre leur partage et leur réutilisation, il a fallu reproduire le fonctionnement des machines physiques au travers des logiciels. C'est cette procédure qui est connu sous le nom de virtualisation. De manière plus formelle, la virtualisation est le moyen par lequel on crée une copie ou version virtuelle d'un équipement informatique. Cet équipement peut être une plateforme physique de calcul, de stockage ou bien une ressource réseau. L'équipement ainsi virtualisé est alors simulé dans un ordinateur et est utilisé comme une entité indépendante. L'infonuagique couplée à la virtualisation ont poussé les opérateurs à repenser le réseau pour le rendre plus flexible, reconfigurable, évolutif, moins sujet aux pannes et moins cher. Ces objectifs ont mené vers la conceptualisation du SDN (Software Defined Networking) et du NFV (Network Function Virtualization).

Le Sofware Defined Networking (SDN) permet de découper les routeurs traditionnels de leur partie décisionnelle. En effet, un routeur traditionnel est constitué de deux plans. Un plan de contrôle qui a pour rôle de calculer les routes sur le réseau en utilisant les algorithmes plus ou moins complexe. Ces routes sont ensuite enregistrées dans une table et permet au routeur de savoir le port de sortie d'un flux qui entre par une interface. Lorsque le flux arrive, le plan de données est chargé de consulter cette table et de faire suivre le trafic sur le port correspondant. L'inconvénient de cette méthode est que la plupart du temps, les décisions du routeur sont fonctions des informations "locales". La notion locale est entre guillemet parce que le routeur a quand même l'état du réseau mais son état global. Dans le technologie SDN, le plan de contrôle est alors centralisé dans un contrôleur qui a une vue globale du réseau et qui décide du comportement du trafic. Le plan de données reste dans le routeur et exécute les décisions obtenues du contrôleur. Cette technologie permet de manipuler le réseau d'un point centralisé. Cette fonctionnalité a comme avantage qu'elle permet une meilleure flexibilité et plus de possibilités dans la gestion d'une requête. Par exemple, avec cette fonctionnalité du SDN, les requêtes venant d'un client A peuvent être traitées conformément à certaines règles et les requêtes d'un autre client B passant par les mêmes équipements réseau peuvent être traitées avec des règles différentes.

Le Network Function Virtualization (NFV) quand a lui permet d'avoir et d'exploiter la version virtuelle des fonctions réseaux. En effet, un équipement réseau contient très souvent une ou plusieurs fonctions réseaux qui y sont intégrées. Par exemple, un routeur CISCO va avoir en plus de la fonction de routage des fonctions de sécurité, de gestion de listes d'accès

etc. Avec l'avènement des fonctions virtuelles, la tendance vers la virtualisation des fonctions réseaux a fait naître le NFV. Il découple la partie matérielle de la partiel logicielle de tous les équipements du réseaux. Cette partie virtuelle peut être un tout, c'est à dire être une parfaite réplique de l'équipement, ou bien peut être un ensemble de plusieurs fonctions indépendantes qui ont été virtualisées.

Le "Service Function Chaining" (SFC) utilise ces deux notions. En effet, le service function chaining ou en français la mise en chaîne des fonctions de service réseau est un suite de fonctions réseaux abstraites par lesquels on fait passer un trafic. Une fonction de service réseau est une fonction qui est responsable d'un traitement spécifique sur les flux qu'elle reçoit. Ces fonctions sont placées dans un ordre précis qui peut changer ou pas en fonctions des paramètres de la chaîne. Tout trafic qui est dirigé dans cette chaîne doit suivre cet ordre. Ces fonctions étant abstraites, elles doivent être assignées à des entités physiques ou virtuelles. Cette assignation est appelé placement de fonction de réseau.

1.3 Problématique

L'arrivée des fonctions virtuelles en chaîne a permis de résoudre un très grand nombre de problèmes. Nadeau et al. (2015) ont décrit les différents problèmes que le réseau traditionnel apporte. Un exemple de ces problèmes est la tolérance aux pannes. Lorsqu'il y a une panne, la plupart des moyens de contournement sont souvent très couteux à mettre en place. De plus, la recherche de la source d'erreur et de la solution au problème prennent du temps. En plus de ce problème, une modification sur le réseau peut devenir très complexe et les avec différentes exigences des utilisateurs et des services de nos jours, le réseau est plus exposé aux problèmes. Les SFC viennent avec l'un des très nombreux avantages de l'infonuagique qui est l'automatisation. En effet, l'erreur humaine est la plus grande source d'erreur sur le réseau de nos jours. Juniper Networks inc. (2008) montre que l'homme est responsable de plus de 50% des pannes sur le réseau. Une étude plus récente confirme (Bednarz (2016)) cela et place la responsabilité humaine à 52%. Donc l'automatisation des processus réduit grandement les possibles erreurs dues à l'action humaine. Pour résoudre tous ces problèmes, Halpern et al. (2015) ont proposé une définition et une architecture pour le SFC (Service function chaining).

Ils ont défini les chaînes de fonctions de service réseau comme "un ensemble ordonné de fonction abstraites avec des contraintes d'ordonnancement qui peuvent s'appliquer sur les paquets, les trames et/ou le flux obtenu après le résultat d'une classification". La classification est une opération qui consiste à ordonner une chaîne de fonction abstraite à l'aide d'un classificateur. Il (le classificateur) peut en plus de cette opération d'ordonnancement proposer

de placer ces fonctions abstraites sur des équipements réels qui peuvent être physiques ou des machines virtuelles. Ces définitions exposent deux grands problèmes inhérents des chaînes de fonction de virtuelles qui sont :

— L'ordonnancement des fonctions.
— Le placement de ces fonctions sur un réseau réel et/ou virtuel.

L'ordonnancement des fonctions et leur placement sont sujets à des coûts financiers ou temporels. En plus, ils peuvent être sujets à des contraintes qui dépendent du réseau, des clients et du flux lui même.

Halpern et al. (2015) décrivent comment une chaîne de fonction doit fonctionner de manière générique. A travers ce RFC, ils établissent les bases de solutions d'ordonnancement et de placement des fonctions de service réseau. D'autres études ont été faites sur ces aspects et les solutions qui sont proposées ont des objectifs bien souvent différents.

— Les coûts (Bouet et al. (2015), Wang et al. (2016) etc.) : ils représentent les dépenses en capital et en opération pour la mise en place et le suivi des chaînes.
— Le délai (Mehraghdam et al. (2014), Ko et al. (2016)) : c'est une des mesures les plus importantes sur le réseau. Il y a principalement deux types de délai : celui généré par le processus de traitement au niveau des fonctions et celui généré par le transport du flux.
— Le SLA (Service level Agreement) (Bari et al. (2015)) : représente les ententes entre les opérateurs de service et les clients (propriétaires de chaîne). Le non-respect d'une règle de ce contrat conduit à des pénalités. L'objectif de ces recherches est la minimisation du non-respect des règles.
— Le nombre de nœuds utilisés (Mehraghdam et al. (2014), etc.) : dans la plupart de ces recherches, les fonctions réseaux sont assignées dans des noeuds que nous appelons centre de données.
— La satisfaction des clients (Ko et al. (2016)) : cette mesure permet de maximiser le degré de satisfaction du client qui est noté en fonction de la violation ou non de certaines de ses contraintes.

Le placement des SFC présente beaucoup de défis.

Ces différents objectifs doivent tenir compte de la position des nœuds sur le réseau, du fait que l'ordre dans les fonctions peut changer ou pas et que à tout moment des nouvelles fonctions peuvent s'ajouter. Le placement ou l'ordonnancement de ces fonctions doit aussi prendre en compte le fait que le débit du flux peut augmenter ou diminuer à n'importe quel moment. En effet, pour accomplir leurs tâches, certaines fonctions peuvent avoir besoin de bourrer les

paquets ou de les modifier (les encodeurs vidéos par exemple). Donc le placement des chaînes de fonctions réseaux et leur ordonnancement doit tenir compte non seulement les nœuds sur lesquels ces fonctions sont assignées mais aussi des liens qui les connectent entre eux.

Le placement des fonctions réseaux est un problème présent dans la littérature. Dans la plupart des textes étudiés dans le chapitre 2, le problème a été trop simplifié. En effet, pour réduire la complexité, les ordres des fonctions sont considérés comme fixe, le flux est conservé entre les fonctions et le trafic est aussi considéré comme inchangé. De plus, les SFC tels que décrit viennent avec un certain nombre de spécification qui ne sont pas toutes prises en compte dans le placement des fonctions réseaux, par exemple : la symétrie.

1.3.1 Le dynamisme

En réalité, le trafic généré par un propriétaire de chaîne varie en fonction des heures de la journée. De plus, depuis l'évolution de l'infonuagique et du développement de l'IaaS (Infrastructure as a service), les méthodes d'assignation des prix des opérateurs de service Internet font que ceux-ci varient en fonction des moments de la journée et de la quantité de données trafic généré. Ce dynamisme créé une nouvelle complexité, car si on place une chaîne de manière optimal à l'heure A, elle pourrait ne plus être optimale à l'heure B. Ceci rend la modélisation du problème plus complexe. Dans les travaux que nous avons explorés, ce problème de dynamisme n'est pas considéré, pourtant il représente un facteur essentiel dans la prise de décision d'une entreprise.

1.3.2 La symétrie

Dans le RFC écrit par Halpern et al. (2015), une des spécificités des chaînes de fonctions est que celles-ci doivent permettre la symétrie. Deux chaînes sont symétriques lorsqu'elles ont une fonction similaire qui doit être placée dans le même centre de données. Cette contrainte complexifie le problème parce qu'elle oblige à avoir des informations sur plusieurs chaînes à la fois.

1.3.3 Les affinités et les incompatibilités

Deux fonctions sont affines lorsqu'elles doivent être placées exactement dans le même centre de données et elles sont incompatibles quand elles doivent être dans deux centres de données différents. Les fonctions peuvent être de la même chaîne ou bien des chaînes différentes. Ce problème survient le plus souvent à cause des raisons de sécurités ou bien des restrictions

liées à la loi. A notre connaissance, nous avons vu juste un travail qui approche partiellement ce problème.

En plus de ces trois gros problèmes, le placement des fonctions doit tenir compte de nombreuses contraintes qui peuvent être la latence, la capacité des centres de données qui changent, les possibilités pour le centre de données de pouvoir héberger certaines fonctions etc. Notre travail se propose de résoudre le placement des chaînes de fonctions en tenant en compte des contraintes et spécifications mentionnées ci-haut.

1.4 Objectif de notre recherche

L'objectif de notre travail est de proposer une méthode pour trouver le placement le moins cher possible des chaînes fonctions virtuelles en tenant compte du dynamisme de la demande des clients et des prix en fonctions des moments de la journée, de la symétrie qu'il peut y avoir entre les chaînes, de l'affinité et de l'incompatibilité entre les fonctions, des contraintes de qualité de service, de capacités et possibilités des centres de données.

Pour atteindre cet objectif, nous avons considéré les hypothèses suivantes :

— L'ordre de la chaîne ne change pas.
— Le flux est conservé.

Nous considérons aussi que les clients peuvent avoir des contraintes supplémentaires, par exemple, le fait que même si la demande change, une fonction n'a pas le droit de changer d'emplacement à une certaine heure.

Le problème est formulé sous forme d'un programme linéaire en nombre entiers avec des contraintes sur les capacités en ressources mémoire, de stockage et de calcul des centres de données. Des heuristiques gloutonne et tabou sont proposées pour améliorer le temps de calcul. Les performances sont comparés avec la résolution exacte du programme linéaire.

1.5 Plan du mémoire

Ce mémoire est organisé de la manière suivante. Le chapitre 2 discute des généralités sur les fonctions virtuelles et fait l'état de l'art en ce qui concerne la recherche sur le placement et l'ordonnancement dans le SFC. Il présente premièrement la virtualisation et les NFV (Network Function Virtualization), ensuite on fait un tour sur le fonctionnement du SDN (Software Defined Networking). A la suite, nous présentons le fonctionnement du SFC et décrivons l'architecture de celui-ci. Enfin, nous ferons l'état de l'art sur le placement des fonctions virtuelles dans lesquels nous ressortirons clairement notre contribution. Le chapitre

3 présente les différents aspects du problème et le modèle mathématique proposé. Le chapitre 4 présente la résolution du problème en utilisant le modèle. Il parle de la génération des paramètres, des ensembles, des conditions de test et présente les résultats. Le chapitre 5 contient les heuristiques gloutonne et Tabou, les résultats de ceux-ci et compare ces résultats avec ceux obtenus par la résolution exacte. Le chapitre 6 conclut notre travail en faisant une synthèse des résultats, présente les limites de celui-ci et fait des propositions d'améliorations.

CHAPITRE 2 GÉNÉRALITÉS ET REVUE DE LITTÉRATURE

2.1 Introduction

La virtualisation, et surtout la virtualisation des fonctions, est en plein essor vu les diverses applications qu'elle offre dans le réseau et l'amélioration de la qualité de la communication. Cette notion a poussé l'évolution du réseau vers la création du "Software Defined Network (SDN)", du "Network Function Virtualization (NFV)" et tout récemment avec l'aide des deux premières des chaînes de fonction réseaux (Service Function Chaining) (SFC). Un grand effort de recherche a été fait dans les SFC ce qui a permis la constitution de son architecture et la proposition des méthodes sous divers critères de placement des fonctions ordonnées ou partiellement ordonnées. Le problème d'assignation des fonctions réseaux peut se présenter sous plusieurs formes et avoir plusieurs objectifs. Ce chapitre présente d'abord un peu plus en profondeur les notions de virtualisation, de SDN et de NFV. Ensuite il parle du SFC de son architecture et de ses défis. Enfin il fait une revue de littérature sur le placement des fonctions réseaux.

2.2 Généralités

2.2.1 Virtualisation

Depuis le début du calcul partagé, la virtualisation a toujours été très présente dans les esprits des concepteurs de réseaux. Comme dit dans Morreale et al. (2014) le concept de virtualisation permet de fournir des services personnalisés aux utilisateurs de ressources en donnant la possibilité à chacun d'entre eux d'avoir le contrôle qu'ils souhaitent sur leurs équipements. L'intégration de ce concept dans le réseau vient réduire les coûts des équipements trop chers, améliorer les logiciels de gestion du réseau et effacer les limites dues à la rigidité de la topologie et à la sous-utilisation des ressources.

Definition

La virtualisation consiste à simuler sous forme logiciel le comportement d'un équipement physique. Il existe plusieurs type de virtualisation et on peut virtualiser beaucoup d'éléments, allant d'une simple mémoire dans une machine à des ressources réseaux complexes.

Virtualisation de la mémoire

La virtualisation de la mémoire est l'une des méthodes les plus connues car les applications ont besoin de plus en plus de mémoire vive. Ce concept consiste à créer une extension de la mémoire dans le disque dur d'un ordinateur.

L'un des gros défis de cette opération est que la mémoire vive a un temps d'accès supérieur au temps d'accès au disque dur. Pour résoudre ce problème lorsqu'on implante la mémoire virtuelle, les informations qui ont besoin d'être sauvegardées et qui sont moins utilisées sont mises sur le disque dur et celles dont on a besoin plus fréquemment sont mises sur la mémoire vive physique.

Virtualisation des serveurs

Permettre à plusieurs personnes d'utiliser les mêmes ressources d'un serveur est le défi qui est à l'origine de la virtualisation des serveurs. Comme explique Morreale et al. (2014), l'idée première est de virtualiser tout l'ordinateur en permettant aux utilisateurs un accès séquentiel aux ressources. Chaque utilisateur a sa machine (virtuelle). Un programme appelé "hyperviseur" a pour but de jouer le rôle de commutateur pour permettre aux différentes machines de partager les ressources séquentiellement.

Dans sa version originelle, un serveur est constitué d'un équipement matériel aux dessus duquel il y a un système d'exploitation. Au-dessus de ce système d'exploitation il y a un ensemble d'application. Le problème avec cet architecture est qu'il ne peut y avoir qu'un seul système à la fois à un instant donné dans un serveur. Ce désavantage est exprimé par le fait qu'un changement du système d'exploitation, si la constitution physique de l'équipement le permet est difficile ; ce qui engendre une inefficacité dans l'exploitation de l'équipement. Avec la virtualisation, on doit mettre un hyperviseur pour faire la jonction entre les différentes machines virtuelles au-dessus et les ressources matérielles en dessous. L'avantage de cette technique est que puisque chaque machine n'est définie que par des fichiers, si elle tombe en panne il suffit de la redémarrer pour la réinitialiser. Si c'est l'hyperviseur qui est en panne, on peut migrer sur un autre serveur les fichiers représentant le système. L'hyperviseur partage principalement les ressources de calcul, de stockage RAM, de réseaux et stockage sur disque dur de la machine physique. Il le fait en considérant les besoins des machines virtuelles et les ressources physiques qu'il a à sa disposition. Donc une meilleure utilisation des ressources.

Type de virtualisation

Il existe plusieurs types de configuration pour aider l'hyperviseur dans sa tâche.

La virtualisation complète : Dans ce type de virtualisation, c'est l'hyperviseur qui est responsable de tout. C'est lui qui simule complètement les ressources physiques en les virtualisant.

La virtualisation complète assistée par l'équipement : dans ce type de virtualisation, les fabricants créent des fonctions primitives dans les processeurs pour permettre d'isoler les machines virtuelles entre elles. L'hyperviseur utilise cette fonctionnalité pour permettre à chaque système virtuel de fonctionner de manière isolée.

Paravirtualisation : dans les types de virtualisation précédentes, les systèmes d'exploitation n'ont aucune information sur le fait qu'elles fonctionnent de manière virtuelle. Avec ce type de virtualisation, ces systèmes sont modifiés pour leur permettre de savoir qu'elles fonctionnent sur un environnement virtuelle.

Virtualisation du système d'exploitation : dans ce modèle, l'hyperviseur n'est pas nécessaire. Chaque machine virtuelle contient des capacités de virtualisation. Un gros désavantage de ce type de virtualisation est que les machines sur une même ressource physique doivent avoir les mêmes système d'exploitation.

Hyperviseur

Un hyperviseur a pour fonction principale de permettre à plusieurs machines virtuelles de tourner sur une même machine physique. Il a pour but d'accueillir ces machines et distribuer les ressources comme il faut entre celles-ci.

Il existe deux grands types d'hyperviseurs :

— Type 1 : ce type est directement relié au matériel de la machine hôte. Il a pour rôle le contrôle du système d'exploitation. Lorsqu'on démarre la machine physique, il prend en charge le contrôle du matériel et alloue l'intégralité des ressources aux machines virtuelles au dessus de lui. Son principal avantage est qu'il permet aux machines virtuelles d'accéder à la quasi-totalité des ressources disponibles. Son principal inconvénient est qu'on ne peut pas installer plusieurs hyperviseur sur la machine physique.

— Type 2 : c'est un logiciel qui s'exécute sur un système d'exploitation déjà présent. Il agit plus comme un émulateur de matériel pour les machines virtuelles qui sont au dessus de lui. Avec ce type, on peut avoir plusieurs hyperviseurs sur la même machine. Par contre il a moins d'efficacité dans l'accès aux ressources.

2.2.2 Software defined networking

Définition du SDN

L'organisme mis en place pour l'étude et la standardisation du SDN est l'Open Networking Foundation (ONF). C'est cet organisme qui a donné la définition la plus complète de SDN. Selon eux, "le SDN est une architecture réseau émergente où la fonctionnalité de contrôle est détachée de celle de commutation et est directement programmable".

Dans cette définition, deux concepts ressortent :

- le découplage de l'architecture en deux fonctions : ceci peut permettre au réseau d'avoir un contrôle plus centralisé et donc permettre de prendre des décisions globales en considérant la vue complète du réseau.

- La programmabilité du réseau : dans ce que nous avons plus haut, une difficulté avec le réseau c'est qu'il dépend des fournisseurs et est rigide. Pour cette raison, lorsqu'une logique a été implantée, si on veut la changer, il faut changer les équipements qui font tourner cette logique et/ou refaire l'architecture. L'idée de programmer le réseau permet à chaque opérateur d'imposer ses propres politiques et leur donne plus de flexibilité dans la décision sur la gestion des paquets.

Avec cette nouvelle vision du réseau, comme expliqué dans Xia et al., 2017, le SDN permet :

- L'amélioration de la configuration dans le réseau : nous savons que la configuration est une fonction très importante dans le réseau et que le fait qu'il y ait plusieurs équipements et de fabricants différents rend cette fonction très délicate et complexe. En unifiant le contrôle du réseau, la configuration est plus facile car on peut configurer une fois et synchroniser cette configuration sur beaucoup d'équipements. L'optimisation d'un réseau peut donc être dynamique et moins complexe.

- le développement de la performance : SDN permet d'avoir une vue globale sur le réseau et donc beaucoup de problèmes de performance reliés à la congestion vont être plus gérables.

- Permettre et promouvoir l'évolution : le réseau étant programmable, et facilement configurable, des nouveaux designs et des nouvelles applications y sont facilement intégrables.

Tous ces avantages soulèvent de gros défis, notamment de concevoir ce type de réseau (SDN) et une fois conçu, le placer avec le réseau existant pour qu'il puisse fonctionner de manière fluide.

Modèle de référence de SDN : architecture

Le modèle architectural du SDN de la figure 2.1 a principalement été proposé par l'ONF. Cette architecture comporte trois couches : la couche infrastructure, la couche de contrôle et la couche d'application. Ces trois couches communiquent entre elle à travers des API.

Couche infrastructure Cette couche contient les équipements de commutation tels les routeurs et les commutateurs. Ces équipements fonctionnent uniquement dans le plan de données. Ils ont généralement deux fonctions. Premièrement, ce sont eux qui collectent les statistiques du réseau, les enregistrent et les envoient au contrôleur pour qu'il puisse les traiter. Deuxièmement ils utilisent les règles imposées par les contrôleurs pour savoir comment traiter le paquet.

Couche contrôleur C'est l'intelligence même du réseau. Il communique avec sa couche inférieure à travers une interface qui est communément appelé en anglais *southbound interface*. Cette interface est utilisée pour permettre au contrôleur de dire quoi faire aux commutateurs qui sont sous sa responsabilité. C'est à travers cette interface que le contrôleur envoie les directives de routage aux infrastructures. Pour la communication avec la couche supérieure, le contrôleur utilise une interface communément appelé en anglais *northbound interface*. C'est à travers cet interface que le contrôleur est informé par exemple de la stratégie de routage. Étant donné qu'il ne peut y avoir un seul contrôleur, car cela pourrait causer un goulot d'étranglement, il est donc avisé de penser à une interface de communication entre contrôleurs.

Couche Application C'est la couche qui implémente les applications et les services présents sur le réseau. Elle est programmable, dynamique et élastique.

Les couches décrites ci-hauts définissent comment se présente le réseau du point de vue SDN. Le placement des fonctions d'une chaîne de réseau dont nous parlerons plus loin est une fonctionnalité qui se placera dans la couche application pour permettre au contrôleur dans la couche contrôleur de diriger le trafic d'une chaîne vers les centres de données concernés.

2.2.3 Virtualisation des fonctions du réseau

Définition

Le concept de NFV nait de la difficulté de créer et de déployer de nouveaux services virtuels sur un réseau physique. Il est aussi le résultat de l'expression du souhait des opérateurs et des acteurs du réseau de réduire les coûts et accélerer le déploiement des fonctionnalités. Il

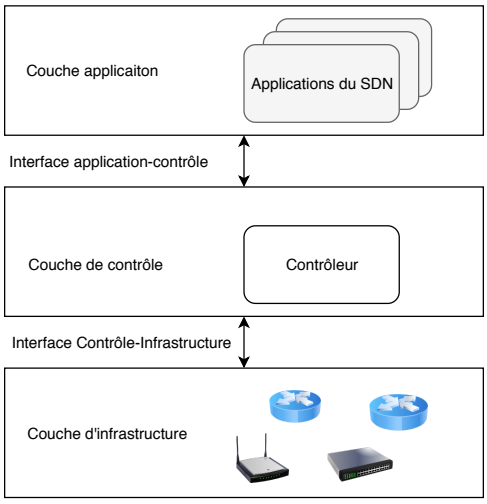


Figure 2.1 Modèle de reférence de SDN

permet ceci en séparant les fonctions comme les pare-feux et les inspecteurs de paquets (DPI) et en les mettant dans des machines virtuelles.

Cette notion permet aux fournisseurs de service d'acheter des serveurs, des commutateurs et des routeurs dans lesquels ils vont faire tourner les machines virtuelles des fonctions réseaux, plutôt que d'investir dans des équipements chers. S'il y a un problème au niveau d'un serveur, ces fonctions peuvent juste être déplacées ou bien relancées ailleurs parmi les autres serveurs du fournisseur. Le NFV permet donc de réduire les dépendances aux matériels, la mise à l'échelle et d'avoir des réseaux très complexes et personnalisés à bas coût.

Pour accelerer le processus de developpement du NFV, l'ETSI (European Telecommunication Standards Institute) a mis sur pied le ETSI ISG NFV (ETSI Industry Specification Group for Network Functions Virtualization) pour définir les requis et les architectures pour virtualiser les fonctions réseaux.

Architecture du NFV

Yi et al. (2018) séparent le NFV en deux couches orchestrées et gérées par une troisième couche appelé MANO (Management and Orchestrator) sur le côté tel que présenté sur la figure 2.2 (inspirée de Yi et al. (2018)).

— La Couche infrastructure : Cette couche est constituée des infrastructures physiques ainsi que le nécessaire en logiciel pour faire tourner les machines virtuelles. Ces infrastructures peuvent être de toutes sortes : points de présences (serveurs, stockage), des routeurs ou des commutateurs.

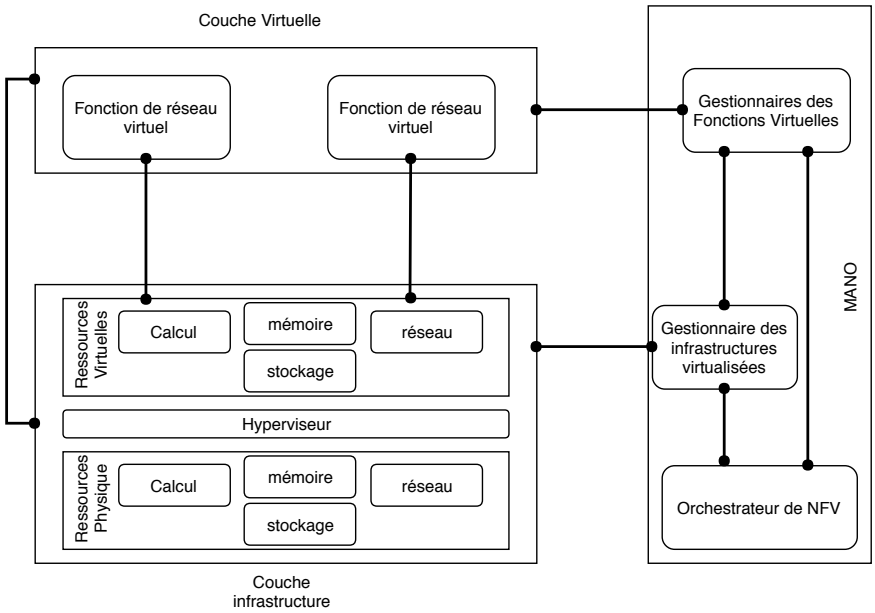


Figure 2.2 Architecture du NFV

— La Couche virtuelle : cette couche au dessus de la couche d'infrastructure contient les fonctions virtualisées qui tournent à l'intérieur des machines virtuelles.
— La Couche de management (MANO) : cette couche agit un peu comme un plan de contrôle. Elle est responsable de l'établissement des connexions entre les fonctions virtuelles et la gestion des ressources au niveau de la couche infrastructure.

2.2.4 Chaine de fonction des services réseau (SFC)

Avec les avancées dans le SDN et le NFV, le besoin est venu de pouvoir créer des réseaux personnalisées sur des infrastructures virtuelles. Ces réseaux comme les réseaux traditionnels sont constitués d'une suite de fonctions ordonnées. Puisque ces fonctions peuvent être lancées sur n'importe quel point de présence, la constitution de ces réseaux et leur placement sur ces points de présence a donné naissance au SFC.

Définition

Le chainage des fonctions de service dans Halpern et al. (2015) est un moyen d'ordonner un ensemble des fonctions de service réseau et de diriger le trafic. Une fonction de service réseau est une boite fermée responsable d'une tâche spécifique sur le réseau. Ces fonctions sont logiques et peuvent avec l'aide des techniques des NFV être assignées à des machines virtuelles ou bien directement à des équipements physiques.

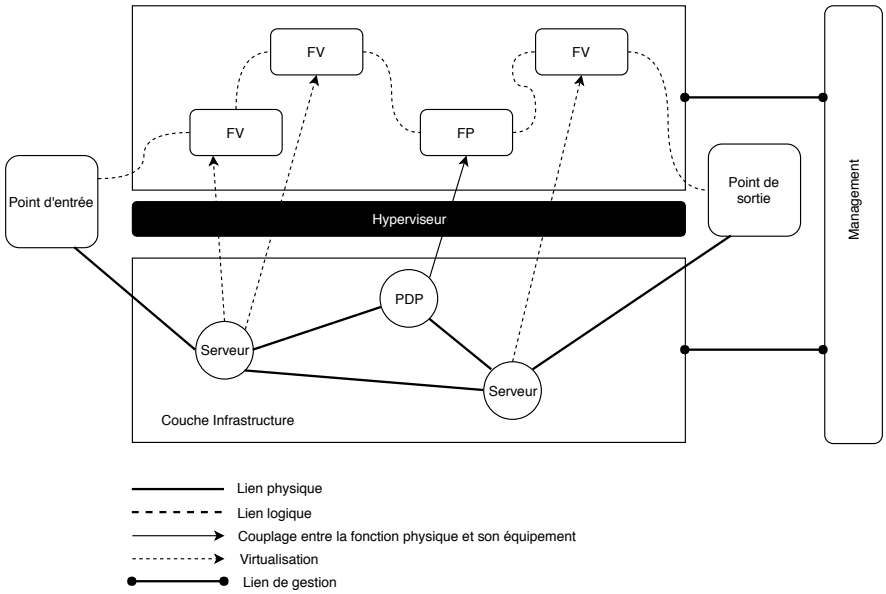


Figure 2.3 Chaine de fonction dans l'architecture du NFV Yi et al. (2018)

Légende pour la figure 2.3

- FV - Fonction Virtuelle
- FP - Fonction Physique
- PDP - Point de présence

La figure 2.3 montre l'intégration d'un chaîne de fonction dans l'architecture du NFV que nous avons présentée plus haut. La chaîne de fonction est formée par les liens logiques. Nous voyons que cette chaîne a un point d'entrée et un point de sortie. Les fonctions virtuelles sont assignées à des serveurs et/ou des points de présence du réseau. Dans cette figure, nous voyons bien le lien entre les fonctions réseaux virtualisées et les SFC. Pour faire fonctionner cela dans la réalité, il faut lui définir une architecture. Cette définition doit tenir compte d'un certain nombre de problèmes.

Les défis de conception de l'architecture du SFC

La définition des standards architecturaux du SFC doit tenir compte d'un certain nombre de défis dont certains sont les suivants :

- La disponibilité et l'élasticité : si le trafic augmente, le réseau doit se reconfigurer et suivre la demande dynamiquement. En effet, cet aspect est très problématique dans les réseaux traditionnels.
- La topologie : les chaînes de fonctions réseaux doivent être indépendantes de la topologie intrinsèque des réseaux physiques qui doivent les contenir.

— La symétrie : qui est un des plus gros défis des réseaux virtualisées. En effet, les SFC doivent permettre des communications bidirectionnelles. Dans une communication bi-directionnel il y a des règles communes au niveau des chaînes qui font que deux chaînes doivent utiliser exactement la ou les même(s) entité(s) virtuelle(s) représentative(s) d'une ou de plusieurs de leurs fonctions communes.
— Le transport : puisque les protocoles et les fonctions sont différents, il faut résoudre le problème d'encapsulation.

Ces défis ont guidé la conception de l'architecture des SFC.

Architecture des fonctions virtuelles

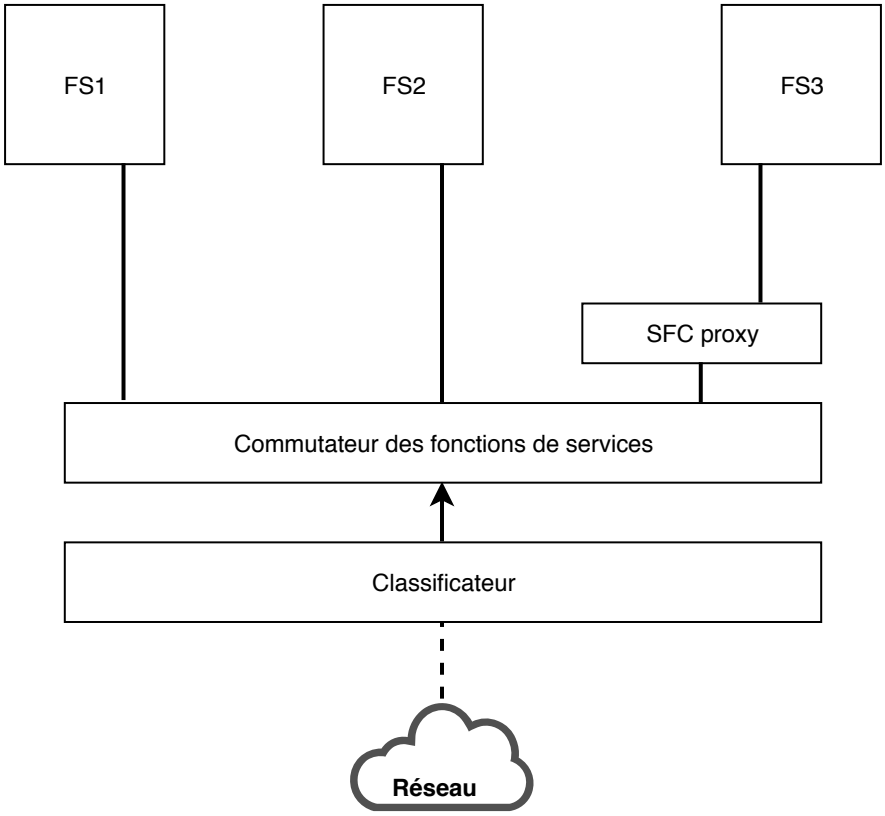


Figure 2.4 Architecture du SFC

La figure 2.4 montre une chaîne de fonction de réseau. En plus des fonctions de service, elle peut avoir des proxy, un ou plusieurs commutateurs de fonction réseau, et un classificateur. Le lien en pointillés entre le réseau et le classificateur contient un flux qui n'a pas été encapsulé. Celui en trait continu contient les flux encapsulés pour la chaîne en question.

— Le classificateur a pour rôle de décider de l'ordre dans lequel les fonctions doivent être

visitées, de faire l'assignation des fonctions logiques sur un réseau réel. Il gère aussi l'encapsulation des informations spécifiques à la chaîne.

— Le commutateur des fonctions du réseau a pour rôle d'envoyer le flux à la fonction suivante. En effet, lorsque l'ordre est établi par le classificateur, le commutateur reçoit le trafic et l'envoie à la première fonction de la chaîne. Une fois le traitement terminé, le trafic est renvoyé au commutateur qui va l'envoyer à la fonction suivante et ainsi de suite. Selon les spécifications du SFC, en fonction du résultat du traitement d'une fonction, les chaînes peut être réordonnées et/ou réassignées. Dans notre travail nous supposons que la chaîne est déjà ordonnée.

— Le proxy. Dans la chaîne de fonction, il y a des services qui sont au courant qu'elles font partie de la chaîne et donc sont capables d'interpréter les informations de contenu dans les données encapsulés. Il y a aussi des fonctions qui ne le sont pas comme la fonction FS3 dans la figure 2.4. Pour être utilisées, ces fonctions ont besoin d'un proxy qui désencapsule les flux qui vont vers elles et les reencapsule à la fin de leur traitement.

Dans notre travail, nous considérons que les commutateurs sont intrinsèques aux centre de données dans lesquels les fonctions vont être assignées, que les encapsulations et les désencapsulations se font de manière intrinsèques et dans un temps négligeable.

2.2.5 Autres concepts importants

Les coûts

Les fournisseurs de service investissent dans l'achat, la mise sur pied et l'exploitation de leurs centres de données. Ces centres de données sont constitués de serveurs donc les coûts varient en fonction du type de processeur, du type et de la capacité des disques durs, des mémoires vives et des cartes mères. En plus de ces serveurs, les fournisseurs de service pour assurer la communication doivent avoir des routeurs et des commutateurs mis en réseau. Ces équipements doivent être refroidis et contenus dans des locaux qui ne sont pas gratuits. De plus, ceux-ci ont une consommation électrique permanente qui doit être payée. Ils sont entretenus par des employés. Ces coûts représentent l'investissement en CAPEX (CApital Expenditure) et en OPEX (Operational EXpenditure) qui dépendent du centre de données. Les investissements OPEX ne sont pas les mêmes à chaque heure de la journée car les équipements ne sont pas utilisés de la même façon à toutes les heures. Leur diversité fait que ces fournisseurs facturent leur service d'hébergement des fonctions et d'utilisation de leur réseau de manière différente à tout moment.

Performances

La performance d'un réseau est définie par plusieurs paramètres, notamment son délai de réponse. Les machines virtuelles dans les centres de données en plus d'avoir pour fonction de traiter et d'envoyer le flux des chaînes sur le réseau réel doivent communiquer entre elles et avec les centres données pour maintenir leur état. De plus, les flux d'information même s'ils sont encapsulés avec des entêtes SFC doivent tout de même être transportés sur des réseaux réels avec des protocoles usuels comme TCP ou UDP. L'utilisation de ces protocoles engendre des délais. Les routeurs entre de les centres de données doivent analyser les paquets pour pouvoir effectuer les opérations de routage. Tous ces traitements vont générer *des délais de transmission* des paquets sur une chaîne.

La mise en marche d'une fonction une fois assignée se fait par le démarrage de la machine virtuelle qui doit la contenir, l'application des configurations dans celle-ci. Ces configurations sont diverses, elles peuvent être des informations reçues par l'encapsulation SFC mais aussi les configurations du réseau virtuel par lequel va passer le flux traité par la machine. Cette mise en fonction génère *un délai de mise en fonction de la chaîne*.

La sécurité et le contexte légal

Les chaînes de fonction peuvent être de toutes formes et transporter tous types d'information. Les utilisateurs de ces chaînes peuvent être dans des banques qui transitent des informations financières sensibles, des hôpitaux qui traitent les données confidentielles de santé de leurs patients, etc. Puisque ces chaînes sont hébergées dans des serveurs communs, il doit y avoir des mécanismes pour empêcher le vol d'information et leur exposition aux personnes qui n'ont le droit d'y avoir accès. Ces mécanismes sont gérées par les fournisseurs de service mais ne sont pas fiable à 100%. Pour cette raison, une inquiétude peut se créer chez les propriétaires des chaînes.

A cause des lois, un flux peut avoir une interdiction de transiter par une région précise. Il peut aussi y avoir des spécificités sur comment les flux doivent être transmis. Ces lois sont différentes en fonction de la zone. Elles vont imposer le type de trafic à faire passer dans un réseau, la qualité de service minimal, les paramètres de sécurité de ce trafic et bien d'autres.

2.3 Revue de littérature

Cette section se divise en deux grandes parties. Premièrement nous allons parler du problème d'intégration de réseau virtuel (Virtual Network Embeeding (VNE)). Nous allons faire res-

sortir les similitudes et les différences avec les problèmes du SFC. Ensuite, nous parlerons des travaux sur le SFC.

2.3.1 Intégration de réseau virtuel (VNE)

Le problème de placement des chaînes de fonction réseau est souvent confondu avec le problème d'intégration de réseau virtuel. En effet, le problème de VNE consiste à assigner à un réseau virtuel les ressources d'un réseau réel. Il alloue à la fois les nœuds du réseau virtuel et les liens de celui-ci sous plusieurs contraintes. On montre dans Fischer et al. (2013) que le problème de VNE est un problème d'optimisation NP-difficile. Ses caractéristiques sont assez similaires au placement des fonctions réseaux dans le cadre du SFC. Mais en plus d'avoir ces caractéristiques, le placement des fonctions réseaux a des contraintes qui le rendent différent. Le tableau 2.1 nous montre deux des grandes différences qu'il y a entre les deux problèmes. La première est que dans le SFC, le trafic doit suivre un ordre prédéfini à l'avance alors que dans le problème du VNE, on a pas vraiment ce genre d'obligation. Deuxièmement dans l'assignation des chaînes de fonctions, on peut réutiliser les machines virtuelles alors que dans le VNE une machine virtuelle ne peut être utilisée par aucun autre nœud. En effet, si on a deux chaînes A et B à placer, lorsque la chaîne A est placée dans une machine virtuelle d'un centre de données, une fonction similaire de la chaîne B peut être placée pour utiliser cette même machine virtuelle, alors que ce n'est pas le cas avec le VNE.

Tableau 2.1 Différence entre le placement des SFC et le VNE

Élement	Placement des SFC	VNE
Ordre	obligatoire	pas obligatoire
Indépendance	Les fonctions sont interdépendantes	Les noeuds sont indépendants

2.3.2 Travaux dans le domaine du SFC

Les chaînes de fonction de service réseaux viennent avec deux principaux défis que nous avons fait comprendre plus haut.

— L'ordonnancement : l'ordonnancement d'une chaîne consiste à trouver le bon ordre dans lequel doit circuler les flux. En général, les fonctions réseaux peuvent transformer le trafic qu'elles reçoivent. Cette transformation peut conduire à augmenter les données ou bien à les réduire. De plus, selon du résultat recherché, il y a des fonctions qui doivent obligatoirement passer avant d'autres dans la chaîne. Ces critères font que l'ordonnancement soit un défi intéressant.

— L'assignation des fonctions dans des points de présence : cette opération consiste à faire héberger une fonction de service dans un point de présence. Lorsque deux fonctions sont dans des points différents, la communication entre les points de présence est un facteur qui doit être pris en compte. De plus, les points de présence ont leur particularité en fonction de la fonction à héberger. Ces détails contribuent à rendre l'assignation des fonctions un autre défi intéressant dans le SFC.

Les travaux sur les chaînes de fonctions de service réseaux sont divers. Certains traitent conjointement l'ordonnancement et le placement des fonctions, d'autres qui ne font que le placement des fonctions. Le tableau 2.2 fait une liste de quelques uns des travaux dans chaque cas. Dans notre recherche nous n'allons pas faire de l'ordonnancement, donc nous allons uniquement nous intéresser aux placements.

Tableau 2.2 Solutions du problème avec ordonnancement vs solutions sans ordonnancement

Avec ordonnancement	Sans ordonnancement
Beck et al. (2017), Mehraghdam et al. (2014), Allybokus et al. (2017)	Bhamare et al. (2017), Addis et al. (2015), Bari et al. (2015), Mechtri et al. (2016) etc.

Classification des recherches précédentes

Les différentes recherches dans le placement des fonctions réseaux suivent des objectifs différents, ont des approches et des priorités différentes et aussi proposent des méthodes de résolution différentes. Les tableaus 2.3 et 2.4 montrent quelques travaux les plus significatifs dans ce sens. Dans le tableau 2.3, la première colonne nous dit de quel article il s'agit, la deuxième nous spécifie les objectifs poursuivis par la recherche et la troisième nous montre avec quelles méthodes ils ont résolus leur problème. Dans le tableau 2.4 la deuxième colonne précise le caractère statique ou dynamique du placement et la dernière nous montre lesquels des paramètres de symétrie, d'affinité ou d'incompatibilité (SAI) ils ont pris en compte.

Le placement de la fonction de service et le chaînage ont trouvé leur origine dans le problème de l'intégration du réseau virtuel. À la fin de 2009, Chowdhury et al. (2009) ont travaillé sur ce problème. L'intérêt principal de cet article était de proposer un algorithme permettant de cartographier les requêtes de fonction de réseau virtuel avec des contraintes de nœuds et de liens. Pour ce faire, ils proposent un programme mixte en nombres entiers (MILP) avec pour objectif de maximiser le gain d'un fournisseur de réseau Internet (InP) en minimisant le nombre de machines utilisées.

Tableau 2.3 Travaux sur le placement des fonctions de services.

Article	Objectif	Solution
Bouet et al. (2015)	Coût	ILP Algorithme glouton
Addis et al. (2015)	Utilisation des liens unité de calcul (CPUs)	MILP
Bari et al. (2015)	SLA	ILP
Wang et al. (2016)	Coût	MILP JoraNFV
Bhamare et al. (2017)	Temps de réponse	ILP ABA
Mechtri et al. (2016)	Nombre de nœud	Eigendecomposition
Allybokus et al. (2017)	Coût	ILP
Ko et al. (2016)	Latence	INLP
Mehraghdam et al. (2014)	Nombre de noeud Latence Débit	MIQCP
Askari et al. (2018)	Latence Blocage Coût	Algorithme heuristique

De nombreuses formulations de ce problème en programme en nombres entiers (ILP) ont été proposées. Comme dans Lin et al. (2015) qui ont présenté dans la couverture de l'IEEE infocom 2015 une étude préliminaire sur le placement des fonctions virtuelles et proposé un modèle ILP pour placer les chaînes de réseaux avec le moins de consommation de ressources. Ils se sont davantage concentrés sur le placement des nœuds mais n'ont pas prêté beaucoup d'attention aux liaisons réseau où le flux passera. Luizelli et al. (2015) ont également proposé un ILP pour placer plusieurs requêtes de fonction de service en même temps dans le but de réduire le nombre d'instances de réseau virtuel utilisées pour assigner sur l'infrastructure. Leur travail par contre n'optimisait pas le chemin utilisé par ces fonctions. Bhamare et al. (2017) ont proposé un modèle de placement des fonctions réseau dans le but de minimiser le temps de réponse total pour les utilisateurs. Dans leur travail, ils ont modélisé le temps d'attente des liaisons comme déterministe avec des arrivées Poissons et des temps de service exponentiel. Ces considérations les aident à calculer le délai des liens et à utiliser ces délais dans un modèle ILP pour placer des chaînes. Tastevin et al. (2017) ont proposé un modèle de programmation en nombres entiers (ILP) pour placer des fonctions virtuelles. Leur ILP a pour but de placer plusieurs requêtes à la fois afin de réduire les coût d'allocation d'une machine virtuelle et de transfert de données sur les liens. Dans leur problème, ils ne considèrent pas un délai de bout en bout pour les requêtes et leurs ressources en liens sont infinies. Ils ne

Tableau 2.4 Travaux sur le placement des fonctions de services.

Article	Dynamisme	Paramètres SAI
Bouet et al. (2015)	Statique	Aucun
Addis et al. (2015)	Statique	Aucun
Bari et al. (2015)	Statique	Aucun
Wang et al. (2016)	Statique	Aucun
Bhamare et al. (2017)	Statique	Aucun
Mechtri et al. (2016)	Partiellement dynamique	Aucun
Allybokus et al. (2017)	Statique	Incompatibilité
Ko et al. (2016)	Statique	Aucun
Mehraghdam et al. (2014)	Statique	Aucun
Askari et al. (2018)	Dynamique	Aucun

contraignent que le point de présence où la fonction virtuelle serait allouée. Bari et al. (2015) ont travaillé sur une formulation ILP pour un problème d'orchestration de fonction de réseau virtuel. Leur travail est divisé en deux parties principales :

- À partir d'un serveur et en fonction de la capacité d'un type de fonction réseau, ils trouvent le nombre d'instances de cette fonction réseau que ce serveur peut héberger.

- Après avoir trouvé le nombre d'instances qu'un serveur peut prendre pour chaque type de fonction réseau, ils placent des demandes de chaînage de fonctions de service. Ce placement consiste à affecter des fonctions réseau d'une chaîne à un serveur sur Internet dans le but de minimiser les coûts d'opération de l'opérateur. En minimisant ces coûts d'opération, ils minimisent la consommation d'énergie, le coût de déploiement du réseau virtuel, le coût de transfert du trafic et le nombre de violations de SLO. Ils ont pris en compte le coût de la location d'une instance sur un serveur et le coût d'utilisation d'un lien physique.

Des modèles non linéaires ont également été présentées. Comme dans Ko et al. (2016) où un problème de programmation non linéaire pour le placement optimal des fonctions de service est proposé. L'objectif du travail était d'optimiser le mécontentement des utilisateurs concernant la latence du service. Dans Mehraghdam et al. (2014), les auteurs ont formalisé des chaînes de fonctions de réseau et ont proposé un programme entier mixte avec des termes quadratiques dans les contraintes (MIQCP) pour placer les fonctions de réseau virtuel. Leur modèle multi-objectif est destiné à utiliser le moins de nœuds de réseau et à minimiser la latence. Au final, ils proposent une analyse pareto qui permet de faire un compromis entre leurs objectifs.

Pour ce qui des algorithmes proposés pour résoudre le problème, Tastevin et al. (2017) ont proposé une heuristique basée sur les graphes divisée en 3 étapes. La première étape consiste

à trouver le nombre minimal N_{min} de point de présence (PDP) à utiliser pour placer toute la requête SFC. La deuxième étape consiste à choisir les N_{min} POP à utiliser. La troisième étape utilise un graphique à plusieurs étapes pour attribuer les fonctions virtuelles et créer le chemin de la requête SFC. Luizelli et al. (2015) proposent une heuristique de recherche binaire dans le but de réduire le nombre de nœuds proposé, où ils ont résolu une version modifiée de leur modèle afin de trouver la meilleure solution possible en peu de temps. Mechtri et al. (2016) ont proposé une méthode quasi analytique pour trouver le meilleur emplacement des chaînes de fonctions virtuelles pour les clients. Cette méthode analytique est basée sur la décomposition en vecteurs propres et présente un avantage considérable par rapport aux autres méthodes car le temps pris par leur heuristique n'est pas fonction du nombre de fonctions virtuelles de la chaîne mais uniquement du nombre de nœuds où ils souhaitent placer ces fonctions. Leur travail est en partie dynamique car il place les demandes les unes après les autres dans le but de réduire les coûts pour l'opérateur. Allybokus et al. (2017) ont proposé un algorithme qui utilise une relaxation de leur modèle ILP pour placer une requête en même temps.

Askari et al. (2018) proposent des algorithmes dynamiques multi-objectifs pour le placement de chaînes de service dans un réseau métropolitain. Ils visent à optimiser l'exigence de qualité de service, la probabilité de blocage des chaînes et les dépenses des opérateurs lors de la mise en place de ces fonctions. Le caractère dynamique de ce travail se traduit dans le fait qu'il place des chaînes qui arrivent dynamiquement à n'importe quel moment dans des nœuds existants.

Analyse et contribution de notre travail

La plupart des travaux mentionnés ci-dessus ne traitent que des contraintes de capacité des nœuds et des liens. Par contre, tel qu'indiqué par Halpern et al. (2015), le chaînage des fonctions de service présente plus de contraintes. Allybokus et al. (2017) ont introduit l'ordre. Étant donné que l'ordre dans la chaîne de service peut changer à tout moment, un placement doit prendre en compte cette possibilité (Halpern et al. (2015)). Allybokus et al. (2017) proposent également une règle anti-affinité où les chaînes non compatibles ne peuvent pas être placées dans les mêmes nœuds. En dehors de ce travail, aucun autre ne propose de placer les chaînes en tenant compte du paramètre d'anti-affinité que nous considérons. La différence avec notre approche de l'affinité et de celle de Allybokus et al. (2017) est que l'anti-affinité qu'il propose est entre les chaînes, alors que nous modélisons l'affinité à l'intérieur des chaînes.

La plus grande partie des travaux que nous avons rencontrés font des placements statiques. Askari et al. (2018) et Mechtri et al. (2016), permettent quand même de recevoir et de placer

les chaînes au fur et à mesure. Par contre, Ils ne tiennent pas compte du fait que les chaînes qu'ils ont déjà placées peuvent avoir des données qui changent et que les centres de données sur lesquels ces chaînes sont placées ont des données qui varient avec le temps. Nous tenons compte de cela dans notre travail. Pour augmenter ce dynamisme, nous avons aussi inclus des restrictions sur les migrations des fonctions qui pourraient avoir besoin d'être assignées.

Lorsque plusieurs chaînes sont placées, aucun des travaux que nous avons rencontrés ne tient compte des liens que ces chaînes peuvent avoir entre eux. Notamment le lien de symétrie que nous avons expliqué plus haut. Notre travail propose d'ajouter et d'étudier cette particularité.

Comme indiqué dans le tableau 2.3, il y a beaucoup de travaux ayant comme objectifs d'améliorer le coût d'investissement. Ce qui concerne les coûts pour les opérateurs. Puisque nous pensons que les personnes qui seront amenés le plus souvent à utiliser les chaînes ne sont pas les fournisseurs de service, nous orientons donc notre travail dans la recherche d'un meilleur gain pour le client propriétaire d'une chaîne.

En résumant, les contributions originales de ce travail sont :

- Au niveau de la modelisation :
  - nous traitons le problème avec beaucoup plus de réalisme que l'état de l'art,
  - l'objectif vise le client,
  - l'introduction de la contrainte de symétrie,
  - contrainte d'affinité entre les fonctions d'une même chaîne,
  - le dynamisme des caractéristiques des centres de données et des chaînes.
- Au niveau de la résolution nous avons conçu un algorithme heuristique tabou ce qui est fait pour la première fois pour résoudre ce genre de problèmes.

2.4 Conclusion

Dans ce chapitre, nous avons fait un survol du concept de virtualisation et du concept du SFC en passant par les concepts intermédiaires comme le SDN et NFV. Nous les avons définis et présenté leurs architectures. Ensuite, nous avons présenté les défis du SFC que nous avons fait suivre par une revue classifiée des travaux sur le placement des fonctions de service. Ces travaux ont été analysés et à partir de cela nous avons sorti notre contribution.

CHAPITRE 3 MODÉLISATION

3.1 Introduction

Le placement des fonctions virtuelles n'est pas un nouveau problème. Comme nous l'avons expliqué au chapitre 2, il a beaucoup de similitude avec le problème du VNE (Virtual Network Embedding). En effet, le placement des chaînes de fonctions réseaux est un problème similaire à celui avec le VNE sur l'aspect "chaîne". Cette chaîne est vue comme un graphe de fonction qui doit être placé dans des centres de données. Au problème de VNE s'ajoute la contrainte d'ordre, la contrainte de réutilisation des fonctions dans le centre de données à quoi s'ajoutent les contraintes évoqués dans Halpern et al. (2015). Donc nous pouvons dire que le placement des chaînes de fonctions réseaux est une extension du problème d'intégration de réseau virtuel. Fischer et al. (2013) montrent que le VNE est un problème NP-complet. Donc le placement des chaînes de fonctions réseaux étant une extension du problème d'intégration de réseau virtuel, est aussi un problème NP-complet.

Ce chapitre porte sur la modélisation comme problème en nombre entiers du placement des chaînes de fonctions réseaux. Nous décrirons en premier lieu le problème ainsi que les ensembles considérés. Ensuite, nous soulignerons les particularités en présentant respectivement la caractéristique dynamique du problème, ensuite les restrictions qu'un client pourrait avoir sur ses chaînes et nous spécifierons les détails sur les types de délais que nous avons considérés. Ensuite, nous présenterons le modèle linéaire et décrirons chaque contrainte, variables et paramètres. Enfin nous ferons une conclusion pour clôturer le chapitre.

3.2 Définition du problème

Le réseau sur lequel nous travaillons est constitué d'un ensemble de nœuds liés entre eux. Tous ces nœuds sont des centres de données et les liens entre eux représentent le chemin pour quitter d'un centre de données vers un autre. Ce chemin est composé de liens physiques, des routeurs et des commutateurs présents sur le réseau. Chaque centre de données a une capacité résiduelle en stockage, en unité de calcul et en mémoire. Cette capacité change chaque fois qu'une chaîne est ajoutée dans ce centre de données. Un centre de données peut être spécialisé pour des types de fonctions particulières. Pour chaque type de fonction qu'il peut avoir, un prix est assigné pour l'hébergement de cette fonction par un client. Ce prix change en fonction de l'heure de la journée. Les liens entre les centres de données ont des délais de transmissions qui changent en fonction des moments de la journée. De plus, faire

passer un flux par un lien a aussi un coût qui dépend de l'heure à laquelle le flux est envoyé. Par exemple, le matin, lorsque les liens ne sont pas très utilisés, le prix pour utiliser un lien peut être très peu élevé pour encourager les utilisateurs à utiliser le réseau dans cette plage. En milieu de journée, ce prix peut atteindre son maximum et recommencer à décroitre dans l'après midi. La particularité de notre approche est que notre système est dynamique et possède des restrictions qui sont dues à ce dynamisme. Nous expliquerons ces notions plus loin dans le document.

3.3 Caractéristiques dynamiques

Nous considérons notre système comme une entité intermédiaire. Il reçoit les requêtes de placement des chaînes de plusieurs client et les place dans différents centres de données qui appartiennent à différents fournisseurs. La figure 3.1 montre comment le système est censé fonctionner. À chaque heure, des clients peuvent envoyer des requêtes de placement de chaîne au système. Toutes ces requêtes sont enregistrées dans une mémoire du système et placées chacune à son tour. A chaque heure de la journée, un client peut demander à retirer sa chaîne du système et un nouveau client peut ajouter la sienne.

Il y a trois types de dynamisme considéré dans notre modèle :

— Le débit
— La capacité
— Le temps

3.3.1 Dynamisme lié au débit

Lorsqu'une chaîne est placée, le système enregistre les positions de chaque fonction de la chaîne. Chaque chaîne a une demande en débit en kbps. Cette demande de flux de données peut changer à chaque heure de la journée. Par exemple supposons que le client C relié à l'hôpital envoie une requête à 8h du matin. A cette heure, il n'y a pas beaucoup de patients dans l'hôpital donc la demande en flux de données n'est pas grande. À 12h, le client a plus de patients. Il a donc besoin d'envoyer plus de données dans le réseau. Sa demande va automatiquement augmenter.

3.3.2 Dynamisme lié à la capacité

Dans le système, il n'y a pas que la demande en flux de données qui changent par heure mais aussi les demandes en capacités. En effet, chaque fonction dans une chaîne a besoin d'une

certaine quantité de ressources en mémoire, en stockage et en unité de calcul (CPU). Lorsque le flux augmente, il est normal de conjecturer que la demande de ces différentes ressources va augmenter à cause du nombre de paquets qui vont s'ajouter.

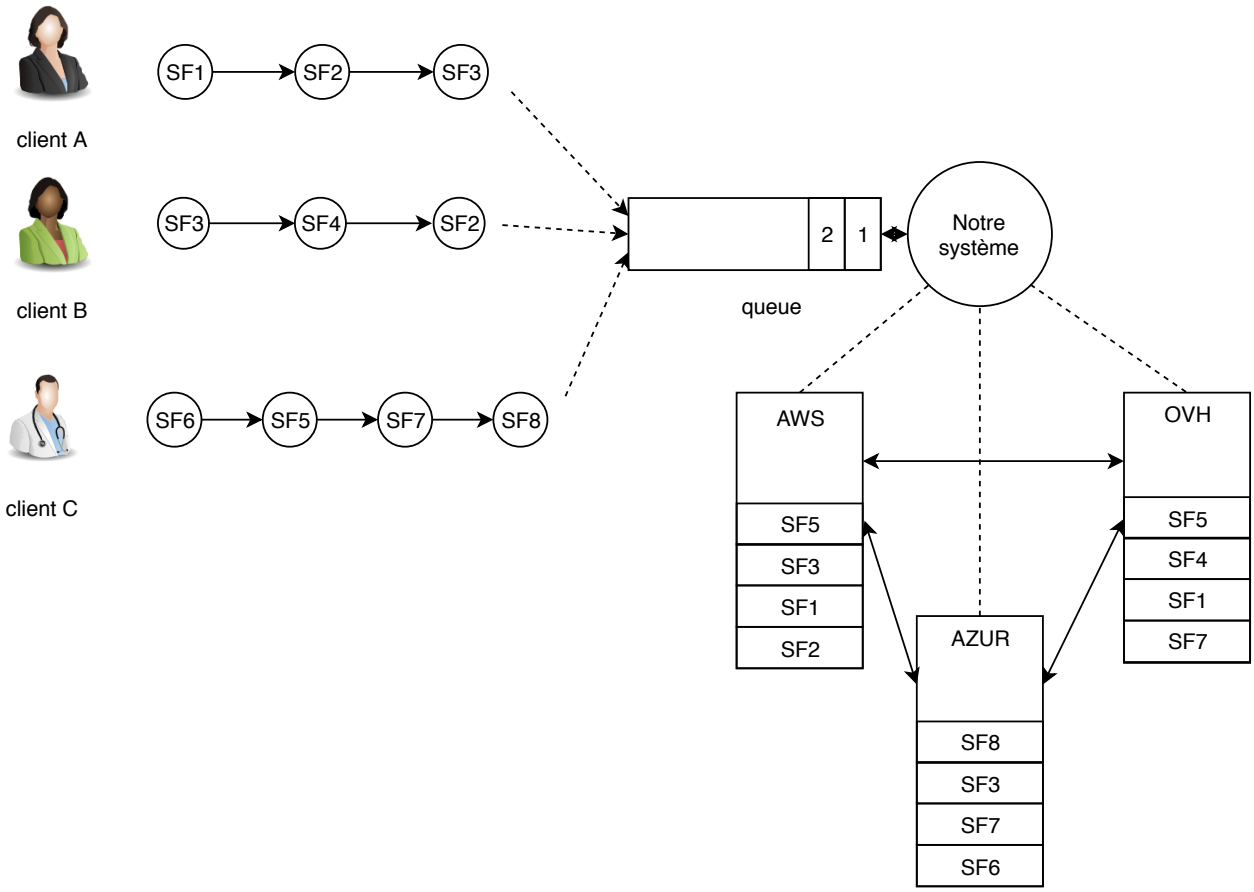


Figure 3.1 Principe de fonctionnement simplifié

3.3.3 Dynamisme lié au temps

Notre système doit placer plusieurs chaînes à chaque heure. En effet, les chaînes placées à une heure qui ne sont pas retirées à l'heure suivante devrons être réassignés à cet heure là. Comme nous pouvons le voir plus bas, certaines chaînes peuvent être liées entre elles. Elles peuvent être symétriques ou incompatibles. Donc placer une chaîne doit tenir compte du placement des chaînes avec lesquelles elle est liée. Pour résoudre cela, notre système doit enregistrer les dernières positions des chaînes déjà placées. Dans ce processus, durant une heure, si une chaîne est placée, son assignation ne peut pas être changée pendant que le système place une autre chaîne. Par exemple, supposons qu'à 8h du matin on doit placer des chaînes A, B et C de notre figure 3.1. La première chaîne que nous plaçons est celle du client A. Le

système enregistre les positions des fonctions de cette chaîne et lorsque nous procédons au placement de la chaîne du client B, aucune des fonctions de la chaîne du client A ne peut être réassignée. Donc le placement de B considère que la chaîne A est déjà placée et que sa position ne change pas. Il en est de même lorsque le placement de la chaîne du client C sera en cours. Les chaînes des clients A et B ne pourront être réassignées. Mais à 9h du matin, lorsque le système doit replacer toutes ces chaînes, si aucune d'elles n'a été retirées par le client propriétaire, ce processus va recommencer dès le début. On va placer la chaîne A sans tenir compte de son placement à l'heure précédente à moins qu'il y ait une règle de non migration sur une de ses fonctions.

3.4 Restrictions

Nous supposons dans notre travail qu'une chaîne a un nœud de départ et un nœud d'arrivée. A l'intérieur de la chaîne, le flux peut être divisé et joint plusieurs fois. Notre but est de placer les requêtes faites par les clients d'une façon dynamique. Cet objectif doit être atteint en tenant en compte de certaines restrictions. Ces restrictions sont énoncées dans des contraintes qui sont définies dans cette suite.

Restriction 1 - Restriction liée au dynamisme en capacité Chaque fonction d'une chaîne particulière a une demande en mémoire, en stockage et en capacité de calcul. Cette demande change dans le temps selon l'utilisation de la fonction dans la journée.

3.4.1 Restriction 2 - la migration

La règle de migration intervient lorsqu'un client veut qu'une ou plusieurs de ses fonctions restent dans le même centre de données lors d'une réassignation à une heure précise. Cette règle peut être présente pendant plusieurs heures et les autres fonctions de la même chaîne qui ne sont pas soumises à cette restriction peuvent changer de centre de données. Par exemple, si la fonction SF1 du client A est assignée au centre de données AWS à 8h, et une instruction de non-migration est mise sur cette fonction, lorsque le système va réassigner cette chaîne à 9h ou plus, cette fonction doit obligatoirement être assignée à AWS jusqu'à ce que la contrainte soit levée ou bien que la chaîne soit retirée du système.

3.4.2 Restriction 3 - le délai

Toutes les chaînes ont un délai maximum permis par le propriétaire de la chaîne. Ce délai traduit la qualité de service que le propriétaire de la chaîne peut avoir. Ce délai peut changer à toutes les heures selon des souhaits du client et du type de chaîne.

3.4.3 Restriction 4 - la symétrie

Des chaînes peuvent être symétriques. Deux chaînes sont symétriques lorsqu'elles ont en commun au moins une fonction et cette fonction doit être placée au même endroit pour les deux chaînes. Dans ce cas, la machine virtuelle ou réelle résultant de l'assignation de cette fonction sera la même pour les deux fonctions.

3.4.4 Restriction 5 - l'incompatibilité

Deux chaînes sont incompatibles lorsqu'elles ont une ou plusieurs fonctions incompatibles. Deux fonctions de deux chaînes différentes sont incompatibles si elles ne peuvent pas être placées dans le même centre de données par exemple pour des raisons de sécurité. Il peut y avoir des fonctions incompatibles dans la même chaîne.

3.4.5 Restriction 6 : l'affinité

Des fonctions peuvent être affines. Dans une même chaîne, le propriétaire de la chaîne peut vouloir que deux fonctions consécutives soient assignées au même centre de données. Ce cas peut arriver lorsque le propriétaire ne veut pas que le flux entre ces deux fonctions traversent le réseau pour des raisons de sécurité ou bien à cause d'une loi, par exemple. Pour simplifier cette contrainte dans notre travail, nous avons supposé que seulement deux fonctions qui se suivent directement peuvent avoir cette contrainte et qu'il n'y a pas de triplet aillant ce type d'affinité. Par exemple dans une chaîne qui sera SF1->SF2->SF3, la fonction SF2 ne peut pas avoir une affinité à la fois avec SF1 et avec SF3.

3.5 Spécification sur les délais - restriction 7

Nous avons considéré plusieurs types de délai dans notre travail. Nous avons parlé plus haut du délai sur les liens. En réalité, lorsqu'une fonction est assignée à un centre de données, il faut la démarrer et la configurer avant qu'elle ne soit active.

Lorsqu'une fonction est assignée, il y a un délai pour la démarrer. Si elle est déjà démarrée, il y a un délai pour la configurer pour une chaîne particulière. Dans ce modèle, nous considérons

que si une fonction est assignée à un centre de données qui a déjà une instance active de cette fonction, le système doit juste configurer cette instance. Si cette fonction est assignée à un centre de données qui n'a pas d'instance active, une instance est alors démarrée et configurée pour que cette fonction soit utilisée. Dans le cas où deux chaînes sont symétriques, il n'y a pas besoin de considérer le démarrage et la configuration de l'instance de la fonction qui doit être ajoutée pour la seconde chaîne.

3.6 Description du modèle ILP

Soit $G(N, E)$ le graphe des différents centres de données et les liens entre eux. $i \in N$ est un centre de données et $(i, j) \in E$ est le lien physique ou logique qui relie les centres de données i et j.

Soit K^h un ensemble ordonné de chaînes qui doivent être placées à l'heure $h \in H$ de la journée, $k \in K^h$. Toutes les chaînes dans cet ensemble sont classées par ordre de placement.

Soit P l'ensemble des fonctions possibles que le système peut offrir.

Soit $G_v(N_v^k, E_v^k)$ le graphe représentant la chaîne de la requête de placement de la chaîne $kinK^h$. Où N_v^k représente l'ensemble des fonctions de service réseau de la chaîne k et est un sous-ensemble de P. E_v^k représente l'ensemble des liens défini sur les fonctions.

Soit $G_v^h(N_v^h, E_v^h)$ le graphe qui contient toutes les chaînes des requêtes de placement que le système a enregistré à $h \in H$. $\bigcup_{k \in K^h} N_v^k = N_v^h$ et $\bigcup_{k \in K^h} E_v^k = E_v^h$

NB : L'indice v est utilisé pour spécifier le caractère virtuel des noeuds et des liens dans les chaînes de fonction de service réseau.

Soit $i \in N_v$ une fonction particulière. Nous définissons donc T_i comme l'ensemble des centres de données qui peuvent héberger cette fonction i.

Soit M_k^h la matrice de placement représentant les placements des chaînes de fonctions précédentes à $k \in K^h$ lorsque nous voulons l'assigner à une heure $h \in H$ de la journée. $(i, n, k_1) \in M_k^h$ avec $i \in P$, $n \in N$ et $k_1 \in K^h / k_1 < k$

Soit $\varphi_{n,l}^{k,h}$ le flux assigné par le système entre les centres de données n et l avant le placement de la chaîne k à l'heure h de la journée, en GigaByte.

Soit H l'ensemble des heures de la journée.

Liens entre les centres de données

A chaque heure, les paramètres de liens entre les centres de données changent.

$D^h_{n,l}$ est le délai en secondes par paquets pour le lien $(n, l) \in E$ à l'heure h de la journée.

Paramètres des liens de la chaîne des fonctions du réseau

d^h_k représente la demande attendue par la chaîne de la requête k à l'heure h de la journée, en GigaByte.

$\overline{D^h_k}$ représente le délai maximum requis par la requête de placement de la chaîne $k \in K$ à l'heure h de la journée, en secondes.

β^h_k représente le délai maximum pour que les instances des fonctions de la chaîne k soient toutes prêtes à être utilisées à l'heure h de la journée. Une instance est prête à être utilisée lorsqu'elle est démarrée et configurée.

Dans la chaîne, le flux peut être divisé à tout moment entre deux ou plusieurs fonctions. Pour cela, nous définissons $\phi^{h,k}_{i,j}$ comme le pourcentage du flux qui va de la fonction i à la fonction j pour la requête de la chaîne k à l'heure h de la journée.

Paramètres des centres de données

En accord avec la restriction 1, nous avons ces paramètres :

C^h_n la capacité de calcul du centre de données $n \in N$ au temps h (en CPUs). Dans ce travail, on assume qu'une unité de calcul est égale à un processeur d'un coeur d'une fréquence de 1GHz.

R^h_n la capacité en mémoire du centre de données $n \in N$ (en MegaBytes) à l'heure h.

S^h_n la capacité en stockage du centre de données $n \in N$ (in MegaBytes) à l'heure h.

Si une fonction est assignée pour la première fois à un centre de données, elle doit démarrer et configurer une instance dans celui-ci. Donc en accord avec la spécification sur le délai, nous avons les paramètres suivants :

- $\delta^{h,1}_{i,n}$ est le délai pour lancer une instance pour la fonction $i \in P$ dans le centre de données n à l'heure h de la journée. Ce délai est exprimé en secondes.
- $\delta^{h,2}_{i,n}$ est le délai pour configurer une instance d'une fonction $i \in P$ dans le centre de données n à l'heure h de la journée. Ce délai est exprimé en secondes.
- $\alpha^{h,k}_{i,n}$ est un paramètre binaire qui indique si une fonction $i \in P$ dans le centre de données n à l'heure h de la journée est active ou pas lorsque le système est entrain de placer la chaîne de la requête k.

Lorsque deux chaînes sont symétriques, les délais de démarrage et de configuration de leur fonction identique ne seront comptés qu'une seule fois.

Paramètres des fonctions réseaux

En partant de la restriction 1, nous avons les paramètres :

$c_i^{h,k}$ est la demande en capacité de calcul de la fonction $i \in N_v^k$ pour la chaîne k à l'heure h, en CPUs.

$r_i^{h,k}$ est la demande en capacité mémoire de la fonction $i \in N_v^k$ pour la chaîne k à l'heure h, en MégaBytes.

$s_i^{h,k}$ est la demande en capacité de stockage de la fonction $i \in N_v^k$ pour la chaîne k à l'heure h, en MégaBytes.

En partant des restrictions 5 et 6, nous avons ces paramètres :

$\gamma_{i,j}^{h,k,1}$ est un paramètre binaire qui indique si une fonction virtuelle i doit être placée dans le même centre de données qu'une autre fonction j à l'heure h dans la chaîne k. Il prend la valeur 1 quand c'est vrai et 0 dans le cas contraire.

$\gamma_{i,j}^{h,k,2}$ est un paramètre binaire indiquant si dans la chaîne k, les fonctions i et j ne peuvent pas être placées sur le même centre de données.

$\zeta_{k,k_1,i,j}^{h}$ est un paramètre binaire indiquant s'il y a des incompatibilités entre la fonction i de la chaîne k et la fonction j de la chaîne k_1 à l'heure h où $k,\ k_1 \in K^h$.

$\eta_{k,k_1,i,j}^{h}$ est un paramètre binaire indiquant si les chaînes k et *k1* sont symétriques et donc les fonctions i de la chaîne k et j de la chaîne k_1 doivent être dans le même centre de données. Avec $k,\ k_1 \in K^h$.

En partant de la restrictions 2, nous avons ce paramètre : $\varepsilon_{i,k}^{h}$ est égale à 1 si la fonction i de la chaîne k ne peut pas être migrée à l'heure h de la journée.

Paramètres de prix

Les centres de données ne sont pas gratuits. Nous considérons que pour chaque fonction, chaque centre de donnée a un prix différent. Ce prix change en fonction de la demande et de l'heure de la journée.

Soit $p_{i,n}^{h}$ le prix pour héberger la fonction virtuelle $i \in P$ dans le centre de données n à l'heure h.

Le trafic dans la plus part des centres de données aujourd'hui est payant chez les fournisseurs d'infrastructures actuelles. Cette tarification dépend aussi des moments de la journée.

Soit $a^h_{n,l}$ le coût d'un GigaByte de trafic entre les centres de données n et l à l'heure h de la journée.

Variables

$$x^{k,h}_{i,n} : \begin{cases} 1, \text{si la fonction virtuelle } i \text{ de la chaîne } k \text{ est assignée au centre} \\ \text{de données } n \text{ à l'heure h}, \qquad \forall i \in N^k_v, n \in N \\ 0, \text{sinon} \end{cases} \tag{3.1}$$

$f^h_{n,l} :\geq 0 \ \forall n, l, h$ la quantité de trafic entre les centres de données n et l à l'heure h en Gb (3.2)

3.6.1 Fonction objectif

Notre problème est de placer les chaînes de fonctions ayant comme fonction objectif la minimisation du coût de l'investissement total de la requête.

$$minimize : \sum_{h \in H} \sum_{i \in N^k_v, n \in N} p^h_{i,n} x^{k,h}_{i,n} + \sum_{h \in H} \sum_{n,l \in N; n \neq l} a^h_{n,l} f^h_{n,l} \tag{3.3}$$

3.6.2 Définitions des contraintes

Contraintes sur les fonctions assignées Chaque fonction d'une chaîne doit être assignée à un et un seul centre de données à la fois ; ce centre de données doit être dans l'ensemble des centres de données qui ont la possibilité d'héberger cette fonction. Si une fonction d'une chaîne a déjà été assignée à un centre de données, lorsqu'on place une autre chaîne cette fonction doit rester dans ce centre de données. Enfin si une fonction n'a pas été assignée à un centre de données, alors elle doit rester non assignée à ce centre de données.

$$\sum_{n \in T_i} x_{i,n}^{k,h} = 1 \qquad \forall i \in N_v^k \; et \; h \in H \tag{3.4}$$

$$\sum_{n \in N - T_i} x_{i,n}^{k,h} = 0 \qquad \forall i \in N_v^k \; et \; h \in H \tag{3.5}$$

$$x_{i,n}^{h,k_1} = 1 \qquad \forall (i,n,k_1) \in M_k^h \text{ avec } k_1 \in K^h \tag{3.6}$$

$$x_{i,n}^{h,k_1} = 0 \qquad \forall i \in N_v, \text{ et } n \in N \text{ et } (i,n,k_1) \notin M_k^h \text{ avec } k_1 \in K^h \tag{3.7}$$

Contrainte sur délai (Restriction 3) On doit s'assurer que le délai maximum de transmission voulu par le propriétaire de la chaîne soit respecté. Cette contrainte s'écrit comme suit :

$$\sum_{(n,l) \in E} D_{n,l}^h \frac{f_{n,l} - \varphi_{n,l}^{h,k}}{d_k^h} \leq \overline{D_k^h} \qquad \forall k \in K \tag{3.8}$$

Contrainte sur le délai de mise en état (Restriction 7 - cas 1) La mise en état d'une chaîne consiste à démarrer les fonctions de cette chaîne et les configurer. Le temps pris par une chaîne pour être mise en état est le temps pris par l'instance de la fonction plus lente à être démarré et/ou configuré. Nous supposons bien entendu que toutes les instances commencent à être démarré et/ou configuré au même moment. On doit s'assurer que le démarrage des instances ne prend pas trop de temps et respecte le délai prévu par le propriétaire de la chaîne. On définit :

$B = \sum_{n \in N} (1 - \alpha_{i,n}^{h,t}) x_{i,n}^{h,k} \delta_{i,n}^{h,1}$ comme étant le délai pour démarrer l'instance dans un centre de données si celle-ci n'est pas encore active.

$O = \sum_{n \in N} x_{i,n}^{h,k} \delta_{i,n}^{h,2}$ sera le délai pour configurer une instance dans un centre de données lorsque une fonction virtuelle lui est assignée.

$Q = \sum_{(j,n,k_1) \in M_k^h / i=j} x_{i,n}^{h,k} \eta_{k,k_1,i,j} \delta_{i,n}^{h,2}$. Comme nous l'avons dit plus haut, lorsqu'une fonction virtuelle est assignée à un centre de données par symétrie, on a pas besoin de relancer ou configurer une nouvelle instance pour elle. Ce terme représente le délai de configuration de la fonction d'une chaîne symétrique si la chaîne à laquelle elle est symétrique est déjà placée.

La contrainte de mise en état est alors :

$$B + O - Q \leq \beta_k^h \qquad \forall k \in K \; et \; i \in N_v^k \tag{3.9}$$

Contrainte de conservation du flux Considérons deux centres de données fictifs s et t qui sont respectivement les centres de données de départ et d'arrivée de toutes les chaînes. Ces centres de données sont crées pour la simplification du problème. Tous les autres centres de données sont reliés à ces deux centres de données avec des coûts de zéro et aucune fonction virtuelle ne peut être assignée à ces centres de données.

La contrainte de conservation de flux sera donc :

$$\sum_{(n,l)\in E|n\in N^h} f_{n,l} - \sum_{(l,q)\in E|q\in N} f_{l,q} = \begin{cases} -\sum\limits_{k\in K} d_k^h \text{ si } l = s \\ 0 \text{ si } n \neq t,\ l \notin \{s,t\}\ q \neq s \\ \sum\limits_{k\in K} d_k^h \text{ si } l = t \end{cases} \tag{3.10}$$

Pour empêcher que des fonctions virtuelles ne soient assignées aux centres de données virtuelles que nous avons créés, nous ne les mettrons pas dans liste des centres de données qui ont la possibilité d'héberger pour chaque fonction possible du système.

Contrainte sur le flux minimum d'un lien Cette contrainte est utilisée pour spécifier que la quantité de flux qui va dans les liens des centre de données doit être au moins égale au flux entre les liens des fonctions qui ont été placées dans chacun des centres de données étant aux bouts de ces liens. Par exemple, dans la figure 3.1, si pour le client A, SF1 est placée dans OVH et SF2 est placée dans AWS, le flux entre OVH et AWS doit être au moins égal au flux entre SF1 et SF2. Dans cette contrainte, $k \in K^h$ représente la chaîne que le système est en train de placer.

$$\sum_{i,j\in N_v^k} x_{i,n}^{h,k} x_{j,l}^{h,k} \phi_{i,j}^{h,k} d_k^h + \varphi_{n,l}^{k,h} \leq f_{n,l} \qquad \forall (n,l) \in E \tag{3.11}$$

Puisque la contrainte est non-linéaire, nous avons besoin de la linéariser. Pour cela, nous introduisons une nouvelle variable binaire y. Cette nouvelle variable dépend des fonctions et des centres de données et est exprimée comme suit :

$$y_{i,j,n,l}^{h,k} = x_{i,n}^{h,k} x_{j,l}^{h,k}$$

Nous ajoutons les contraintes suivantes qui nous permettrons de connaitre les valeurs y :

$$y^{h,k}_{i,j,n,l} \leq x^{h,k}_{i,n} \tag{3.12}$$

$$y^{h,k}_{i,j,n,l} \leq x^{h,k}_{j,l} \tag{3.13}$$

$$x^{h,k}_{i,l} + x^{h,k}_{j,l} - 1 \leq y^{h,k}_{i,j,n,l} \qquad \forall\ (i,j) \in E^k_v \text{ et } (n,l) \in E \tag{3.14}$$

La contrainte de flux minimum sur les liens devient alors :

$$\sum_{(i,j)\in E^k_v} y^{h,k}_{i,j,n,l} \phi^{h,k}_{i,j} d^h_k + \varphi^{k,h}_{n,l} \leq f_{n,l} \qquad \forall (n,l) \in E \text{ et } h \in H \tag{3.15}$$

Contraintes d'affinité et de non-affinité (Restriction 6) Pour des raisons de sécurité ou légales, le propriétaire d'une chaîne peut vouloir que deux fonctions qui se suivent dans sa chaîne soit assignées au même centre de données. Cette contrainte s'écrit comme suit :

$$x^{h,k}_{i,n} \geq \gamma^{h,k,1}_{i,j} x^{h,k}_{j,n} \qquad \forall i,j \in N^k_v,\ n \in N, \text{ et } h \in H \tag{3.16}$$

Dans la même chaîne, il peut y avoir des fonctions qui doivent être dans des centres de données différents. Cette contrainte s'écrit comme suit :

$$x^{h,k}_{i,n} + \gamma^{h,k,2}_{i,j} x^{h,k}_{j,n} \leq 1 \qquad \forall (i,j) \in E^k_v,\ n \in N, \text{ et } h \in H\ k\ in\ K^h \tag{3.17}$$

Contrainte d'incompatibilités (Restriction 5) Il peut y avoir des incompatibilités entre chaînes. Donc les fonctions concernées par cette incompatibilité ne peuvent pas être dans le même centre de données.

$$x^{h,k}_{i,n} + \zeta^h_{k,k_1,i,j} x^{h,k_1}_{j,n} \leq 1 \qquad \forall i \in N^k_v \text{ et } j \in N^{k_1}_v,\ n \in N, \text{ et } h \in H\ k, k_1\ \in\ K^h \tag{3.18}$$

Contrainte de migration (Restriction 2) La contrainte de migration est là pour éviter qu'une fonction ne soit déplacée lorsqu'on veut la réassigner. Elle s'écrit telle que suit :

$$x_{i,n}^{h+1,k} \geq \varepsilon_{i,k}^{h+1} x_{i,n}^{h,k} \forall h \in H,\ n \in N,\ i \in N_v^k \tag{3.19}$$

Contraintes de symétrie et de capacités des centres de données (Restrictions 1 et 4) Si deux fonctions de deux chaînes différentes sont symétriques, le système doit les placer dans le même centre de données. Cette contrainte est écrite comme suit :

$$x_{i,n}^{h,k} \geq \eta_{k,k_1,i,j} \qquad \forall (j,n,k_1) \in M_k^h,\ \text{et } i \in N_v^k \text{ et } h \in H \text{ et } k_1 \in K^h \tag{3.20}$$

Pour être sûr qu'il y a de la capacité disponible lorsqu'on place des chaînes qui ont leur chaîne symétrique déjà placée, nous supposons que l'information sur la chaîne que nous devons placer est déjà présente dans le système. Donc nous devons être certains en plaçant la première chaîne qu'il y a assez de place dans le centre de données pour l'autre chaîne. Considérons $\iota = (\eta_{k,k_1,i,j} = 1) \wedge (j,k_1,n) \notin M_k^h$ avec $k,\ k_1 \in K^h$. Ce terme permet d'avoir les chaînes symétriques à la chaîne qu'on est entrain de placer qui ne sont pas encore assignées donc qui ne sont pas dans la matrice de placement.

$$\sum_{i \in N_v^k} x_{i,n}^{h,k} c_i^{h,k} + \sum_{k_1 \in K-k} \sum_{\substack{i \in N_v^k, \\ j \in N_v^{k_1}/\iota}} c_j^{h,k_1} x_{i,n}^{h,k} \leq C_n^h - \sum_{(i,n,k) \in M_k^h} x_{i,n}^{h,k} c_i^{h,k} \qquad \forall n \in N \tag{3.21}$$

$$\sum_{i \in N_v^k} x_{i,n}^{h,k} r_i^{h,k} + \sum_{k_1 \in K-k} \sum_{\substack{i \in N_v^k \\ j \in N_v^{k_1}/\iota}} r_j^{h,k_1} x_{i,n}^{h,k} \leq R_n^h - \sum_{(i,n,k) \in M_k^h} x_{i,n}^{h,k} r_i^{h,k} \qquad \forall n \in N \tag{3.22}$$

$$\sum_{i \in N_v^k} x_{i,n}^{h,k} s_i^{h,k} + \sum_{k_1 \in K-k} \sum_{\substack{i \in N_v^k \\ j \in N_v^{k_1}/\iota}} s_j^{h,k_1} x_{i,n}^{h,k} \leq S_n^h - \sum_{(i,n,k) \in M_k^h} x_{i,n}^{h,k} s_i^{h,k} \qquad \forall n \in N \tag{3.23}$$

3.7 Conclusion

Dans ce chapitre, nous avons présenté le modèle que nous avons utilisé pour résoudre le placement des chaînes en tenant compte du changement de la demande en terme de flux, de capacité de temps et de prix, la contrainte de symétrie et la contrainte d'affinité. Nous avons deux variables principales qui représentent respectivement l'assignation d'une fonction d'une chaîne à un centre de données et le flux entre deux centres de données. Pour pouvoir atteindre nos objectifs, nous avons ajouté une troisième variable qui est le produit de la variable

d'assignation et avons linéarisé le problème. Enfin nous avons considéré que les informations sur la symétrie, la compatibilité et l'affinité soit connues au moment du placement de la première chaîne concernée. Dans le chapitre suivant, nous allons montrer les résultats de calcul de ce modèle à la suite de quoi nous proposerons une approche heuristique pour faire ce placement.

CHAPITRE 4 RÉSOLUTION EXACTE

4.1 Introduction

Pour résoudre les problèmes NP-complets, il y a plusieurs méthodes qui s'offrent à nous :

— La résolution exacte : cette approche permet d'avoir une solution optimal.
— La résolution heuristique : elle permet d'avoir une solution assez bonne dans un temps raisonnable lorsque la résolution exacte n'est pas possible (i.e. grandes instances du problème).

Dans ce chapitre, nous allons exploiter le modèle présenté dans le chapitre précédent pour résoudre à l'optimalité notre problème dans différents cas de figure. Nous allons premièrement présenter notre système de génération de données. Nous entendons par cela les principes et les lois sur lesquels nous nous sommes basés pour la génération de nos différents paramètres. Nous décrirons l'architecture du système et comment il produit ces paramètres. À la suite de cette partie, nous établirons les ensembles de tests que nous avons générés et montrerons les temps de résolution de ces tests. Enfin nous évaluerons certains de ces cas pour ressortir les performances et les dépendances du système en fonction des contraintes qui rendent notre travail différent de celui des autres.

4.2 Caractéristique du système de génération des données

4.2.1 Mise en place

Pour pouvoir mettre en place notre résolution exacte, nous avons écrit le modèle de programmation linéaire en nombre entier que nous avons proposé en nous servant du langage de modélisation AMPL. La résolution de ce modèle utilise la version 12 de l'outil d'optimisation Cplex-Studio. Notre système de génération des données a été écrit avec le langage Python version 3.6.

Nous avons choisi AMPL et Python à cause leur simplicité. De plus, Python étant un langage interprété, il permet de faire des tests rapidement et simplement parce que nous n'avons pas besoin d'une compilation préalable.

4.2.2 Le dynamisme

Notre système est sensé faire le placement de manière dynamique des fonctions de réseaux. L'aspect dynamique vient du fait que les demandes en terme de capacité et de flux varient dans le temps. En plus de ces demandes, les paramètres des liens entre centres de données et des centres de données eux mêmes sont aussi variables. Nous avons créé alors trois périodes de la journée :

- Une période de pointe : pendant cette période, les demandes sont à leur plus haut niveau et les prix aussi sont plus hauts que la normale. Cette période se situe entre 10h et 20h avec une pointe autour de 15h.
- Une période normale : durant cette période, les demandes sont normales, ne varient pas beaucoup et le prix sont raisonnables. Elle se situe entre 4h du matin et 10h du matin.
- Une période creuse : cette période caractérise les plus petites demandes. Pour inciter les chaînes à être utilisées, les opérateurs proposent des prix très bas. Elle va de 20h dans la soirée à 4h du matin le lendemain.

Les capacités, les demandes et les délais changent en fonction de ces périodes. Ces changements seront expliqués plus en détails dans la suite du travail.

4.2.3 Topologie du réseau

Le réseau est composée de centre de données et de liens. On définit un lien comme une connexion entre deux centres de données. Nous considérons que la connexion entre deux centres données peut passer par Internet donc notre lien peut être constituée en réalité non seulement d'une seul lien physique mais aussi d'une suite de routeurs, de commutateurs et tout autre équipement pouvant intervenir dans la communication entre deux entités sur Internet.

Délai entre le centre de données

Tous les liens ont un délai qui dépend des routeurs et des nœuds d'accès par lequel ils passent. Ces délais sont générés aléatoirement entre deux valeurs pas très éloignées. La valeur ainsi générée est multipliée par un coefficient qui dépend de la période de la journée à laquelle l'on se trouve (voir tableau 4.1).

Dans nos tests, nous allons varier entre cinq et quarante le nombre de centres de données. Nous avons fait une liste de dix-huit fonctions réseaux possibles. Les chaînes et les centres de données se construisent en se basant sur ces dix-huit fonctions (voir tableau 4.2).

Tableau 4.1 Liste des coefficients représentatif de la période.

Période	Coefficient
Heures de pointes	1.5
Heures normales	1
Heures creuses	0.7

Tableau 4.2 Liste des fonctions de service.

Sigle	Signification
FWD	Pare feu
WAN	Service pour les grands réseaux
AppAcc	Accelerateur d'application
SLB	Distributeur de charge au niveau des serveurs
NAT44	Traducteur d'adresse réseau 44
NAT64	Traducteur d'adresse réseau 64
HINJECT	Fonction d'injection d'identifiant d'hôte
HHeader	Fonction d'enrichissement de l'entête HTTP
TCPOpt	Optimiseur du protocole TCP
IntSys	Système de détection des intrusions
PreSys	Système de prévention des intrusions
VPN	Fonction de réseau privé virtuel
DHCP	Serveur de gestion du protocole de configuration dynamique des hotes
IPAud	Audit IP
PROINS	Inspecteur de protocole
WEBOPTCTL	Controleur de l'optimisation web
APPFWD	Pare feu d'application
DPI	Inspecteur de paquet en profondeur

Génération des chaînes

Les chaînes sont générées de manière aléatoire et restent inchangées tant qu'elles sont dans le système. Une chaîne compte entre 3 et 8 fonctions. Avant la douzième heure de la journée, aucune chaîne n'est retirée du système. Tant que le nombre maximum de chaînes spécifié dans le fichier configuration n'est pas atteint, une nouvelle chaîne est ajoutée et le nombre de chaînes qui sont ajoutées à une heure précise est aléatoirement choisit entre 1 et le nombre de chaînes restantes. Après cette douzième heure, les chaînes peuvent être retirées du système avec une probabilité de 10%. Une nouvelle chaîne peut être ajoutée avec une probabilité de 18%. Lorsque l'évènement de retrait d'une chaîne arrive, nous avons choisi de retirer les chaînes les plus anciennes avant les chaînes les plus récentes du système. Lorsque notre système n'a plus qu'une seule chaîne, il est impossible de la supprimer jusqu'à la fin de la

journée. Les chaînes peuvent contenir des branches parallèles, ces paramètres sont générés pendant la génération de la chaîne que nous présenterons plus loin.

4.2.4 Charge de travail dynamique

La charge de chaque chaîne varie comme nous l'avons expliqué plus haut en fonction de l'heure. Pour simuler les périodes de pointes, normales et creuses, nous avons utilisé une distribution gaussienne dont la valeur médiane reste la même mais les déviations changent dépendamment de la période dans laquelle l'on se trouve. Le tableau 4.3 montre les différentes déviations choisies.

Tableau 4.3 Liste des déviations gaussiennes représentatives de la période.

Période	Déviations
Heures de pointes	20%
Heures normales	50 %
Heures creuses	90 %

4.2.5 Génération des centres de données

Le nombre de centre données est spécifié par l'utilisateur dans un fichier de configuration à cet effet.

Les capacités des centres données

Les capacités de chaque centre de données sont générées aléatoirement. Un centre de données est caractérisé par le nombre d'unités processeur qu'il contient, sa quantité de mémoire vive et enfin sa capacité en stockage permanent. Une unité processeur est égal à 1GHz. Le nombre d'unités processeur pour le calcul est choisi aléatoirement entre 150 GHz et 350 GHz. Bien entendu nous savons que dans la réalité il n'existe pas de serveur ayant de telles caractéristiques. Nous voyons cela comme la somme des capacités de toutes les machines qui constituent le centre de données. De même, la quantité de mémoire vive varie entre 64 Gb et 1000 Gb et la capacité de stockage entre 150 Gb et 10000 Gb.

Délais de lancement, de configuration et de transmission

Chaque centre de données pour une fonction particulière prend un temps à démarrer et un temps pour configurer la fonction. Le temps de lancement d'une fonction est choisi aléatoi-

rement entre 30 et 100 ms et le temps de configuration est entre 3 et 10 ms. Ces temps ne changent pas en fonction des instants de la journée.

Le délai de transmission entre les centres de données a une composante aléatoire qui varie entre 10ms et 35ms à toute heure. Cette valeur est multipliée par les coefficients du tableau 4.1.

Fonctions possibles

Les fonctions que les centres de données ont la possibilité d'héberger sont choisies au hasard dans un pool de 90% ou 50% de fonctions possibles.

4.2.6 Le coût d'hébergement et d'utilisation des liens

Le prix est l'un des paramètres dynamiques de notre système. Le prix dépend à la fois du centre de données de l'heure et du type de fonction. Dans notre système, les prix pour héberger des fonctions réseaux sont générés aléatoirement et choisis entre 0.1 et 1.2$. Ce prix est soumis aux mêmes coefficients que le délai de transmission. Ainsi, d'après le tableau 4.1, dans les heures de pointes ce prix est multiplié par 1.5.

Le prix du lien entre les centres de données varie aussi mais n'est pas dépendant du coefficient de la table 4.1. Il est aléatoirement généré entre 0.005 et 0.25 $/Gb aux heures normales, entre 0.15 et 1$/Gb aux heures de pointe et enfin entre 0.025 et 0.1$/Gb aux heures creuses. Le coût du lien entre le centre de données A et le centre de données B n'est pas le même que celui du centre de données B vers le centre de données A parce qu'ils peuvent passer par des chemins différents.

4.2.7 Détails de la génération des chaînes

Chaque chaîne a un nombre de fonctions qui est décidé dans le fichier de configuration. Chaque fonction d'une chaîne a un besoin en capacité processeur, en unité de calcul et en unité de stockage. Le besoin en qualité de service des chaînes est exprimé par les exigences en terme de délai de transmission total et du délai de lancement de la fonction la plus lente à mettre en place.

Demande en capacité

Les besoins en capacité des fonctions sont générés aléatoirement à chaque heure. Le besoin en unités de processeur est généré entre 1 et 3 Ghz, celui de la mémoire vive entre 1 et 4 Gb

et celui du stockage entre 0 et 2Gb. Ces besoins sont aussi multipliés par les coefficients de périodes présentés au tableau 4.1.

Limites de delai

Les limites en terme de délai sont aussi générées aléatoirement. Les délais de lancement et de configuration varient dans le temps mais ne dépendent pas de la période. Le délai de mise en service est constitué du délai de lancement et de configuration et est noté β dans le modèle linéaire du chapitre 3. Il a été choisi entre 500 et 1000 ms chaque heure. Le délai de transmission est choisi entre 500 et 2000 secondes et est multiplié par les coefficients du tableau 4.1 en fonction de la période de la journée dans laquelle on se trouve.

Restrictions de symétrie, d'affinité, de migration et d'incompatibilité

Puisque l'une de nos principales contributions de ce travail est l'addition des contraintes de symétrie, d'affinité et d'incompatibilité, nous avons généré les paramètres nécessaires. Deux fonctions de deux chaînes différentes ont 13% de chance d'être symétriques si elles ont une fonction similaire entre elles. Si elles en ont une deuxième, cette probabilité va baisser à 6%. Nous avons fait ceci pour que des chaînes n'aient pas trop de symétrie ce qui pourrait augmenter le taux de rejet. Donc dans notre système, une chaîne a très souvent un maximum de deux fonctions symétriques.

Pour ce qui est de la contrainte d'incompatibilité, la probabilité d'avoir une incompatibilité entre deux chaînes est de 10%.

Nous avons aussi introduit la contrainte d'affinité : seules les fonctions liées directement dans la chaîne peuvent avoir cette contrainte. Pour des raisons de simplicité, une fonction ne peut pas avoir deux ou plus fonctions affines dans la même chaîne. Le caractère affine de deux fonctions arrive avec une probabilité de 10% dans une chaîne.

La migration empêche le système de re-allouer une fonction à un autre centre de données. Cette contrainte est caractérisée par un paramètre de migration qui permet de savoir si à un heure précise une fonction doit rester dans son centre données ou non. Ce paramètre est généré indépendamment de la chaîne ou bien du type de fonction. Une fonction se voit imposer de ne pas migrer avec une probabilité de 10%.

Branche parallèle à l'intérieur de chaîne

Comme nous l'avons expliqué dans le chapitre précédent (3), une chaîne peut avoir des branches parallèles. Au point d'entrée de ces branches, le trafic doit se diviser et se reconstituer au point de sortie. La quantité de trafic qui va dans chaque branche au point d'entrée est obtenu aléatoirement entre 10 et 100% du trafic. Nous nous sommes néamoins assurés que la quantité de trafic à la sortie soit égale à celle d'entrée.

4.3 Comportement dynamique du système et ensemble des tests

Comme nous l'avons présenté dans le chapitre 3, les chaînes seront traités les unes à la suite des autres. La raison principale est que les chaînes n'arrivent pas forcément au même moment et que nous avons besoin d'informations sur le placement de certaines chaînes pour en placer d'autres. De plus, nous avons pour objectif de réduire individuellement le coût d'assignation de chaque chaîne.

4.3.1 Comportement dynamique du système et architecture

Notre modèle comporte deux grands types de paramètres :

- Les paramètres statiques : ils sont générés avant le lancement de la résolution AMPL. Exemple : la capacité des centres de données, la symétrie entre les chaînes ou encore les affinités entre les fonctions d'une chaîne.
- Les paramètres dynamiques : ils sont régénérés à chaque fois qu'une chaîne est placée. Exemple : la matrice placement des fonctions précédentes.

La symétrie entre les chaînes est générée dans les paramètres statiques alors que les chaînes arrivent dynamiquement. En effet, parce que nous voulions éviter les rejets de placement des chaînes, nous avons supposé que si deux chaînes sont symétriques, alors avant le placement d'une chaîne on a déjà les informations sur la deuxième chaîne. Ceci devra servir à faire une "réservation" de ressources pour la chaîne suivante. Ceci a été bien appliqué dans la contrainte sur les capacités présentés dans le chapitre 3. Dans la suite, nous verrons l'architecture dynamique de fonctionnement de notre système et nous décrirons les étapes d'exécution de celui-ci.

Architecture du fonctionnement du système

La figure 4.1 nous montre exactement comment le système de génération de données est sensé fonctionner.

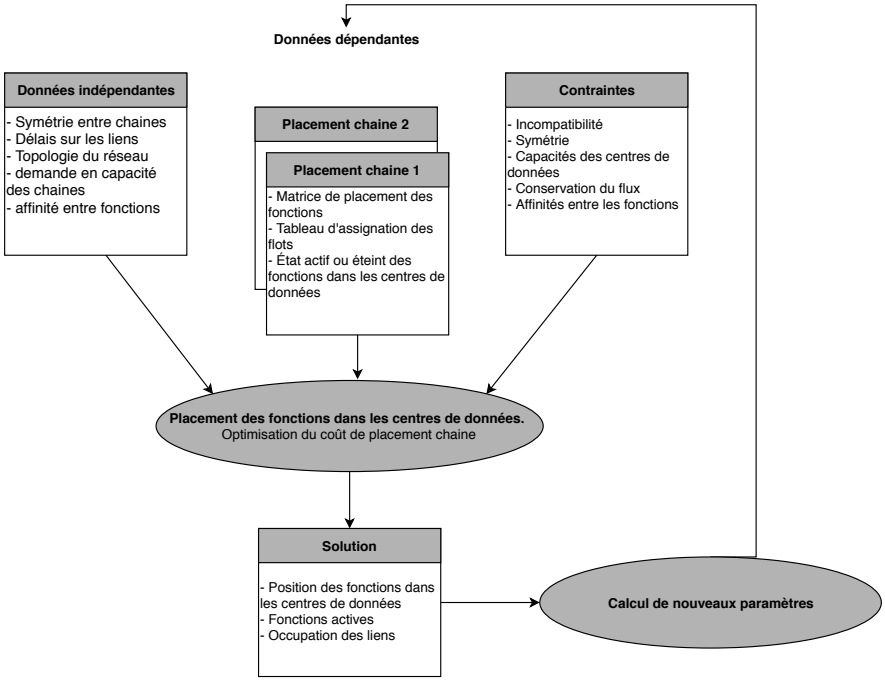


Figure 4.1 Architecture de fonctionnement du système

Les étapes de fonctionnement de notre système sont les suivantes :

Étape 1 - Génération des paramètres statiques et des données des chaînes Nous faisons la planification sur une journée. Pour cela, nous générons les informations des centres de données sur cette période en respectant les règles établies dans la section précédente. Ensuite, nous générons les chaînes avec la planification de leur apparition au cours de la période. Les paramètres de ces chaînes comme les informations qui les lient (la symétrie et l'incompatibilité) sont aussi générées durant cette étape. Une fois généré, un fichier est créé avant de passer à l'étape suivante.

Étape 2 - Génération des paramètres dynamiques Les paramètres dynamiques sont pour la plupart obtenus après exploitation des résultats du placement précédent. Lors de l'assignation de la première chaîne, ces paramètres sont tous mis à zéro et les ensembles générés dynamiquement sont assimilés à l'ensemble vide ($\emptyset$). Lorsqu'il y a déjà eu au moins un placement, le programme exploite respectivement la position des fonctions placées et les flux sur les liens entre les centres de données pour générer les matrices de placements, de flot et les informations sur les fonctions actives dans les centres de données. Une fois ces paramètres générés, le système choisit la prochaîne chaîne à placer et passe à l'étape suivante.

Étape 3 - Assignation des chaînes avec AMPL Dans cette étape, le système prend en entrée les paramètres statiques, les paramètres dynamiques et le fichier de représentation du modèle avec le langage AMPL pour pouvoir placer les fonctions. Ce fichier contient les spécifications du modèle ainsi que toutes les contraintes de celui-ci. Ces informations sont envoyées au solveur cplex-studio pour faire le calcul. Le résultat est enregistré et exploité à l'étape suivante.

Étape 4 - Extraction du réseau et retour à l'étape deux Dans cette étape, le résultat obtenu est mis dans une forme compréhensible par le système et si toutes les chaînes ont été placées alors il arrête la résolution et présente le résultat, sinon il retourne à l'étape 2 de génération des résultats dynamiques.

4.4 Ensemble de tests et temps de résolution

Pour juger si notre modèle et notre système de génération des données et d'exploitation de résultat fonctionne, nous avons défini un ensemble de tests. Cet ensemble est conçu non seulement pour tester la fonctionnalité mais aussi l'efficacité, les limites, et l'impact des contraintes les unes par rapport aux autres sur l'ensemble des solutions. Les tableaux 4.5 et 4.4 montrent les ensembles des tests en fonctions des variations sur le nombre de fonctions, le nombre de centre de données et les pourcentages du nombre de fonctions possibles dans les centres de données.

Légende pour les tableaux 4.5 et 4.4

- CD - Centre de données
- NF - Nombre de fonctions
- NC - Nombre de chaînes
- % CDF - Pourcentage des fonctions possibles dans les centres de données.

Les tableaux 4.5 et 4.4 nous permettent de voir comment se comporte la résolution du modèle lorsque nous augmentons le nombre de fonction et le nombre de centre données. Nous avons pris les deux cas où le nombre de fonctions possibles dans un centre données s'élèvent à 50% des fonctions disponibles (4.5) et où le nombre de fonction s'élèvent à 90% des fonctions disponibles (4.4). Dans la première colonne nous avons le nombre de centre de données, ensuite nous avons le nombre de fonctions que les chaînes contiennent. La troisième colonne montre les temps de résolutions de chaque test et la dernière colonne représente les gaps AMPL-Cplex. Nous constatons que Cplex prend moins de temps à résoudre le problème dans le cas à 50% que dans le cas à 90%. Ce qui nous permet de conclure que plus les centres

Tableau 4.4 Ensemble des tests ainsi que les temps moyens de leur résolution et les gaps AMPL : cas où CDF = 90%

CD	NF	Temps de résolution (s)	Gaps Ampl (%)
5	3	0.03	0
5	5	0.06	0
5	8	0.08	0
8	3	0.11	0
8	5	0.18	0
8	8	2.56	0
10	3	0.14	0
10	5	0.42	0
10	8	1.07	0
15	3	0.55	0
15	5	8.96	0
15	8	56.60	0.005
20	3	3.17	0
20	5	20.28	0
20	8	121.38	0.01
25	3	9.9	0
25	5	53.95	0
25	8	140.11	0.006
30	3	5.25	0
30	5	68.43	0
30	8	173.76	0.50
40	8	pas assez de mémoire	

de données sont spécialisés plus petit est le temps de résolution. Nous constatons aussi que après quarante (40) centres de données, le système arrive à saturation et Cplex se trouve à court de mémoire pour la résolution du problème.

Le nombre de fonctions a une grande influence sur le temps de calcul.

4.5 Étude de cas

Les contraintes de notre problème n'ont pas le même effet sur le coût total de placement, sur le taux d'utilisation des centres de données et sur le taux de rejet des chaînes. Dans cette partie, nous montrons comment la solution fonctionne, nous évaluons les contraintes et ressortons les limites de notre solution.

Tableau 4.5 Ensemble des tests ainsi que les temps moyens de leur résolution et les gaps AMPL : cas où CDF = 50%

CD	NF	Temps de résolution (s)	Gaps AMPL (%)
5	3	0.01	0
5	5	0.03	0
5	8	0.08	0
8	3	0.08	0
8	5	0.12	0
8	8	0.09	0
10	3	0.10	0
10	5	0.13	0
10	8	2.56	0.029
15	3	0.45	0
15	5	1.77	0
15	8	27.17	0.066
20	3	0.20	0
20	5	6.17	0
20	8	46.09	
25	3	2.40	0
25	5	30.29	0
25	8	437.98	0
30	3	2.37	0
30	5	31.09	0.048
30	8	91.63	0.06
40	3	4.75	0
40	5	55.43	0
40	8	pas assez de mémoire	

4.5.1 Résolution du problème

Cas du placement d'une chaîne

Dans le placement d'une chaîne, plusieurs facteurs peuvent être observés. Les principaux sont le changement de centre données d'une fonction en fonction de l'heure et du prix de celui-ci ou bien en fonction de l'affinité entre fonctions à une heure précise. La figure 4.2 montre le placement d'une chaîne qui est constituée de huit fonctions : Nat64, Hheader, ProIns, TcpOpt, IpAud, DPI, FWD et AppAcc. Elle doit être placée dans 5 centres de données au choix.

Dans la figure 4.2, les fonctions de la chaîne sont représentées par des cercles noirs. A l'intérieur de ces cercles sont inscrits les noms des fonctions. Ces fonctions sont connectées entre

elles par des flèches qui indiquent le sens du flux. Les centres de données sont représentés par des carrés noires à l'intérieur desquels sont inscrits leurs noms. Pour des raisons de lisibilité, nous n'avons pas mis les liens entre les centres de données qui sont connectés deux à deux pour former un graphe complet.

A l'heure 1, les fonctions NAT64 et HHEADER sont affines et doivent être placées dans le même centre de données. Entre zero heure et deux heures, les paramètre des centre de données et des chaînes ne changent pas en dehors de cette affinité. Nous observons bien qu'à cause de cette affinité, la fonction HHEADER est forcée de se placer avec sa fonction affine dans le centre de données DIK. Cette fonction revient à son centre de données normale à l'heure 2. Elle va rester assignée là jusqu'à l'heure 5 où le centre de données DIK va se révéler être plus avantageux que le centre de données GIB. Nous remarquons aussi que le centre de données DAJ n'est utilisé qu'une seule fois à l'heure 3 parce que son coût améliore le placement de la chaîne à la fonction FWD. À cette heure, le centre de données CEB qui contenait les fonctions FWD et APPACC n'est plus utilisé par ces deux fonctions car il engendre un plus grand coût.

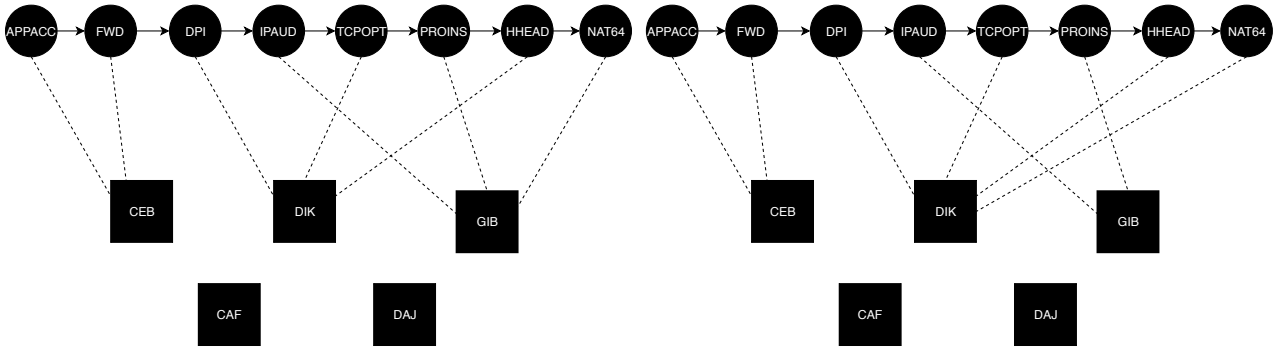
Placement de deux chaînes

Dans cette partie, nous plaçons deux chaînes. Nous allons observer la symétrie, l'incompatibilité et la migration dans ce cas. Les deux chaînes que nous plaçons ont toutes cinq fonctions. La première chaîne en noir sur la figure 4.3 contient les fonctions : TcpOpt, AppAcc, PreSys, Wan, HHeader. La deuxième chaîne en blanc contient les fonctions : IpAud, AppAcc, DPI, TcpOpt, AppFWD. Elles vont être placées dans 5 centres données qui sont réprésentés par des carrées noirs.

La figure 4.3 nous montre le placement de la première chaîne. Comme nous pouvons le constater la deuxième chaîne n'est pas encore placée. Ceci se justifie par le fait que les chaînes arrivent chacune à son tour et que pendant les deux premières heures de la journée, la deuxième chaîne n'est pas encore présente. Cette chaîne arrive à l'heure 2 tel que nous le voyons dans la figure 4.4.

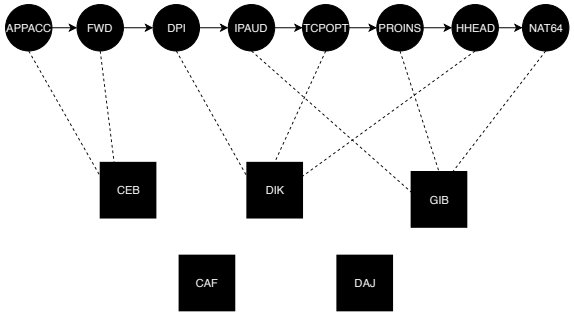
A l'heure 2, les chaînes que nous plaçons sont symétriques. La fonction concernée par cette symétrie est la fonction TcpOpt. Nous constatons que la fonction dans les deux chaînes est assignée au centre de données DED. Cette assignation change à l'heure 3 quand la contrainte de symétrie se lève.

Les fonctions IpAud de la chaîne 2 et AppAcc de la chaîne 1 sont incompatibles aux heures 2 et 3. Elles ne sont par conséquent pas placées dans les mêmes centres de données.

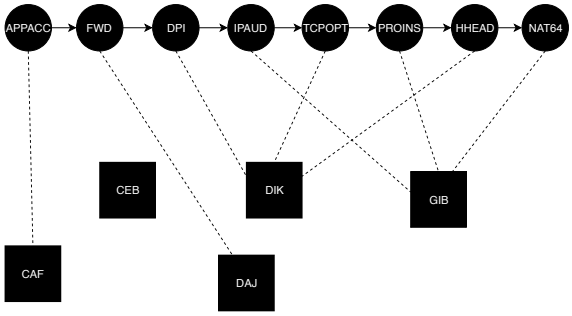


(a) 00 :00

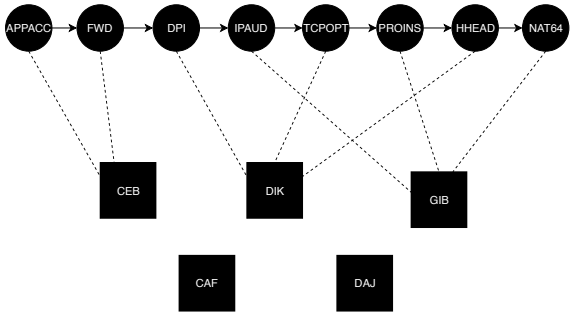
(b) 01 :00



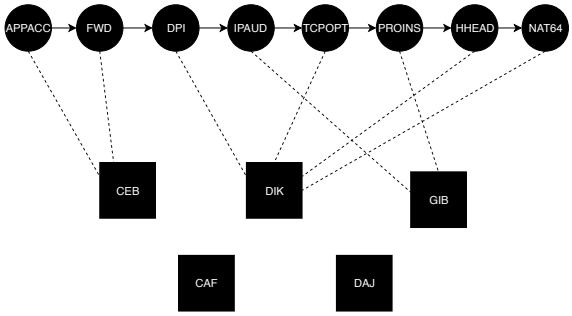
(c) 02 :00



(d) 03 :00

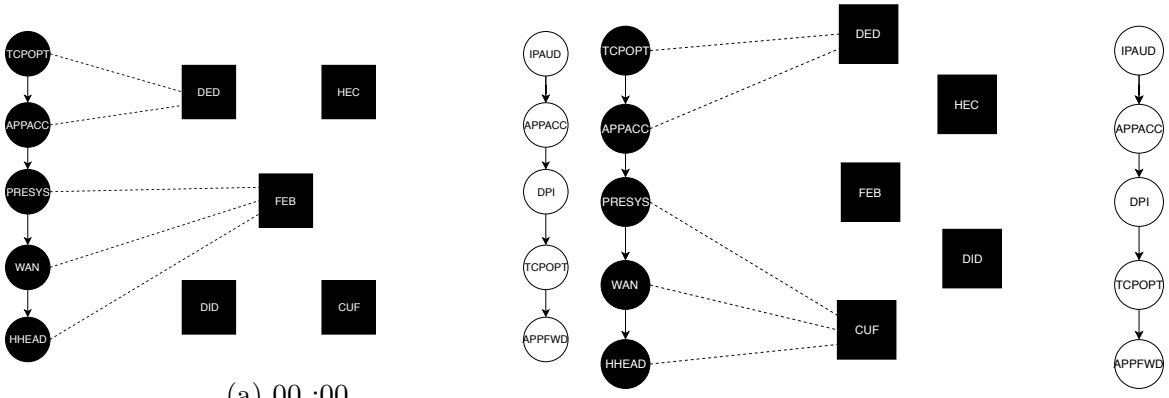


(e) 04 :00



(f) 05 :00

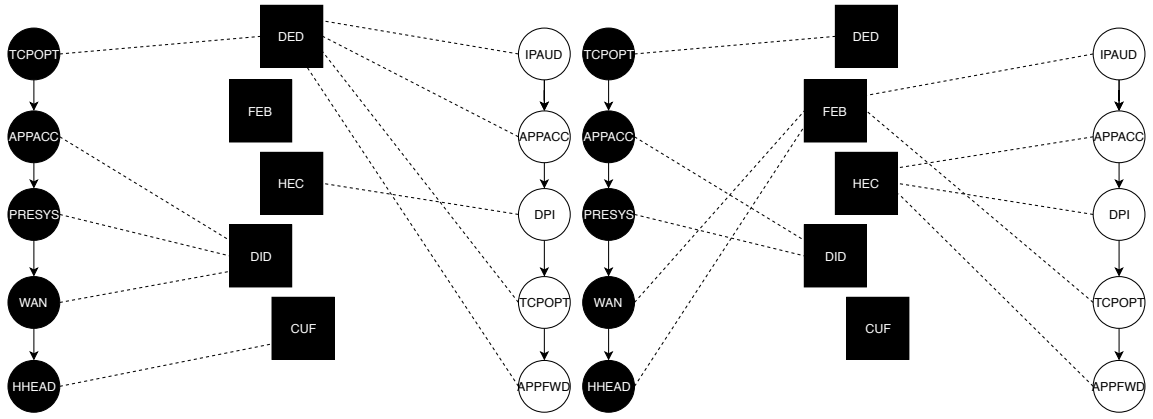
Figure 4.2 Placement d'une chaîne



(a) 00 :00

(b) 01 :00

Figure 4.3 Placement de deux chaînes : chaîne 2 pas encore présente



(a) 02 :00

(b) 03 :00

Figure 4.4 Placement de deux chaînes : incompatibilité et symétrie

La figure 4.5 nous montre comment la migration est interdite. La fonction AppAcc de la chaîne numéro une ne doit pas migrer à 20 :00. Nous constatons qu'à 19 :00 cette fonction est assignée au centre de données DID et ne change pas de centre données à l'heure suivante. Mais à 21 :00, cette contrainte est levée et la fonction est alors assignée au centre de données DED.

Dans les tests avec plusieurs chaînes, il arrive que des chaînes soient rejetées. L'une des principales causes de ce rejet est le fait que deux fonctions soient symétriques et qu'une fonction de la chaîne en cours de placement soit incompatible avec une fonction d'une chaîne qui est déjà placée et qui se trouve dans le même centre de données que sa fonction symétrique.

4.5.2 Comparaison des contraintes

L'ajout des contraintes dans un problème d'optimisation va engendrer une augmentation du coût. Nous allons dans cette sous section voir l'augmentation ou la diminution du coût en fonction de la contrainte d'affinité, d'incompatibilité, de symétrie et de migration.

Nous avons séparé les contraintes en deux groupes.

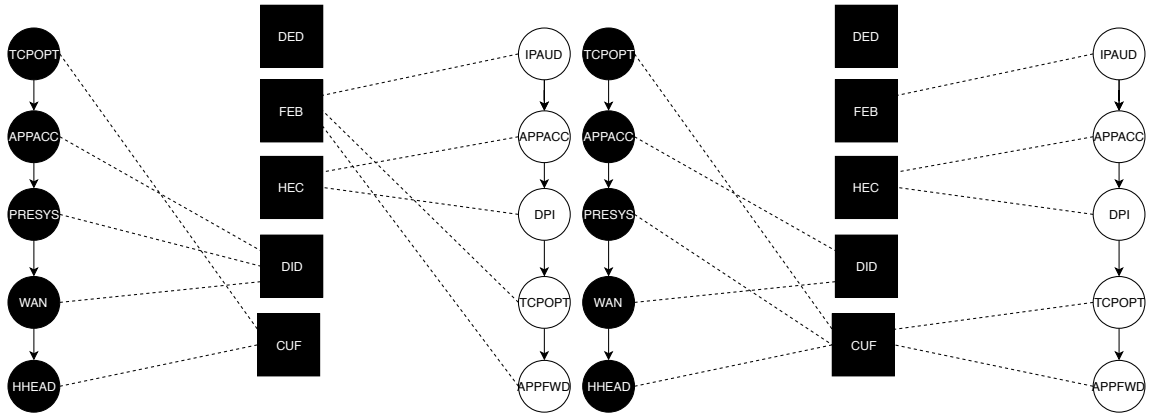
- Les contraintes intra-chaînes : il s'agit des contraintes qu'il y a sur une même chaîne. Par exemple, la contrainte d'affinité ne dépend d'aucune chaîne autre que la chaîne en cours de placement. La contrainte de migration non plus ne subit pas l'influence d'une autre chaîne.
- Les contraintes inter-chaînes : il s'agit des contraintes entre deux chaînes. Ces contraintes sont la symétrie et l'incompatibilité.

Nous allons comparer les contraintes de chaque groupe entre elles.

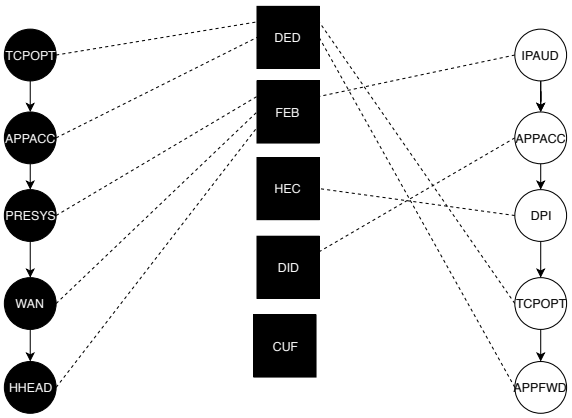
Affinité et migration

Le tableau 4.6 nous montre l'impact respectif de l'affinité et de la migration sur le coût final d'assignation de fonction. Pour obtenir ces valeurs, à cause du caractère aléatoire de la génération de données, chaque couple (nombre de centres de données, nombre de fonctions) a été exécuté 10 fois et nous avons retenu la moyenne de ces résultats dans ce tableau.

Dans le tableau 4.6 la première colonne représente le nombre de centres données utilisés pour le test, la deuxième le nombre de fonction des chaînes placées, la troisième colonne représente la différence relative obtenue entre l'objectif du problème avec la contrainte affinité (OAA) et l'objectif sans la contrainte d'affinité (OSA). Enfin la quatrième colonne représente la différence relative obtenue entre l'objectif du problème avec la contrainte de migration (OAM) et l'objectif sans la contrainte de migration (OSM). Ces valeurs représentent le pourcentage

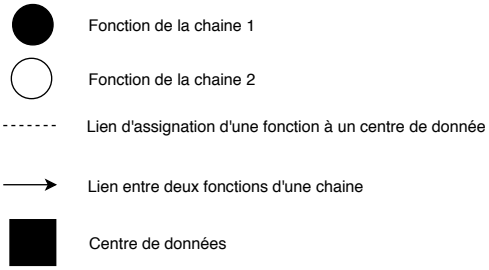


(a) 19 :00 (b) 20 :00



(c) 21 :00

Figure 4.5 Placement de deux chaînes : migration



de la différence entre ces différents objectifs. Pour avoir ceux-ci les formules suivantes sont utilisés :

— $Différence\ Affinité\ =\ \frac{OAA-OSA}{OAA} \times 100$

— $Différence\ Migration\ =\ \frac{OAM-OSM}{OAM} \times 100$

D'après les résulats du tableau (4.6), nous constatons que dans les chaînes de fonction contenant 5 fonctions, la migration a dans deux des cas plus d'impact sur le coût que l'affinité entre les fonctions. Ceci peut s'expliquer par le fait que dans des chaînes de 5 fonctions il est moins probable d'avoir des fonctions affines que dans des chaînes ayant plus de fonctions. Dans le reste des cas, nous constatons que l'affinité entre les fonctions coûte plus cher que la migration. Plus le nombre de centre de données est grand plus les différences des coûts causés de ces contraintes est grand. Le coût supplémentaire peut aller jusqu'à 10 %

Tableau 4.6 Comparaison de l'affinité avec la migration

Centre de données	Fonctions	Diff. Affinité (%)	Diff. migration (%)
3	5	0.06	0.04
3	8	0.08	0.02
6	5	2.43	3.12
6	8	2.22	1.49
10	5	0.48	2.57
10	8	9.96	5.74

Comparaison de l'incompatibitité avec la symétrie

Le tableau 4.7 nous montre les impacts des contraintes de symétrie et d'incompatibilité sur l'objectif final.

Dans le tableau 4.7 la première colonne représente le nombre de centre données utilisés pour les tests, la deuxième, le nombre de fonction des chaînes placées. La troisième colonne représente la différence relative obtenue entre l'objectif du problème avec la contrainte d'incompatibilité (OAI) et l'objectif sans la contrainte d'incompatibilité (OSI). Enfin la quatrième colonne représente la différence relative obtenue entre l'objectif du problème avec la contrainte de symétrie (OAS) et l'objectif sans la contrainte de symétrie (OSS). Ces différences sont obtenues à partir des expressions suivantes :

— $Différence\ Incompatibilité\ =\ \frac{OAI-OSI}{OAI} \times 100$

— $Différence\ Symétrie\ =\ \frac{OAS-OSS}{OAS} \times 100$

Nous constatons que l'incompatibilité revient comme étant la contrainte la plus couteuse des deux. En effet, la contrainte de symétrie n'a pas un grand impact sur le coût. Son impact sur des petites et moyens réseaux est insignifiant. Ceci nous pouvons l'expliquer par le fait

que sauf en cas d'incompatibilité, une fonction identique sur deux chaînes aura tendance à se placer dans le même centre de données et ce peu importe s'il y a symétrie ou pas entre ces deux chaînes.

L'incompatibilité par rapport à toutes les quatre contraintes est la plus couteuse. Nous observons cela dans pratiquement tous les cas.

Tableau 4.7 Comparaison entre l'incompatibilité et la symétrie.

Centre de données	Fonctions	Diff. incompatibilité (%)	Diff. symétrie (%)
3	5	8.91	0
3	8	4.55	0
6	5	5.12	0
6	8	12.8	0.01
10	5	156	0.01
10	8	9.69×10^{-1}	0

4.5.3 Taux de rejet

À cause des contraintes inter-chaînes, nous avons souvent remarqué le rejet des chaînes. Tel que illustré dans la figure 4.6, sur un placement de 100 chaînes successives, nous avons constaté que lorsque le pourcentage de fonction que peut héberger un centre de données est supérieur à 75%, le taux de rejet est inférieur à 5%.

En dessous de ce pourcentage jusqu'à 50%, le taux de rejet va être 7% lorsqu'il devient 70%, autour de 15% lorsqu'il est de 60%. Il augmente et peut arriver des fois jusqu'à 20%. Dans 73% des cas, le rejet est causé par l'incompatibilité entre les chaînes et le fait qu'il y ait pas beaucoup d'autres opportunités pour le placement.

Pour améliorer ce taux une plus grande diversité au niveau du nombre de centre de données est bénéfique.

4.6 Conclusion

Dans ce chapitre, nous avons présenté le fonctionnement de notre modèle. Nous avons tout d'abord décrit le système de génération des données. Nous avons ensuite décrit comment ces données sont utilisées pour placer les chaînes dynamiquement. Après cela, nous avons présenté les ensembles qui nous ont servi pour les tests. Pour ces ensembles, nous avons évalué les temps de résolutions ainsi que les gaps de résolution avec AMPL-Cplex. De plus, nous avons illustré des exemples de placement des chaînes. Nous avons fait deux cas, un dans

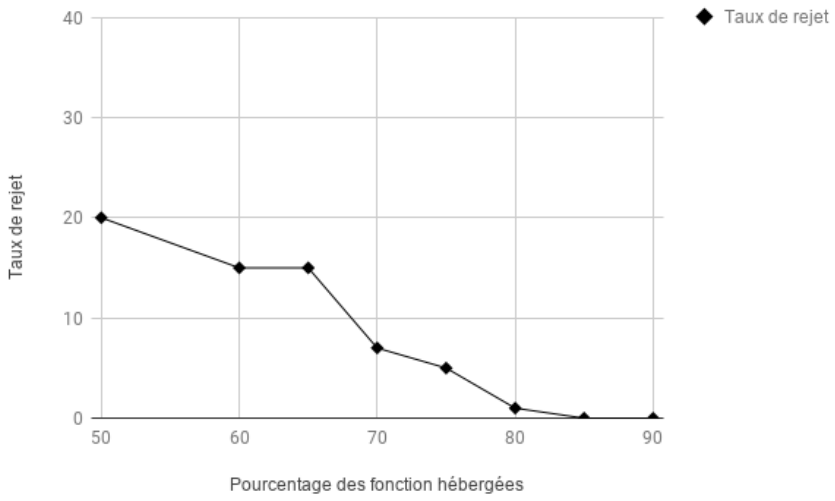


Figure 4.6 Taux de rejet par rapport au pourcentage des fonctions hébergées dans les centres de données

lequel on place un chaîne et un autre dans lequel on place deux chaînes. Cette illustration nous a permis de voir l'application de la symétrie, de l'incompatibilité, de l'affinité et de la migration dans le placement de ces chaînes. Nous avons en plus de cela fait une comparaison entre les contraintes pour voir laquelle est la plus coûteuse pour la fonction objectif. Il en ressort globalement que la contrainte de symétrie est très peu couteuse pour l'objectif en comparaison aux autres contraintes. Il est globalement plus couteux d'avoir des fonctions affines que d'imposer la non migration d'une fonction surtout lorsque le nombre de fonction dans la chaîne devient grand et que le nombre de centre de données devient important.

CHAPITRE 5 HEURISTIQUES

5.1 Introduction

Dans la plupart des problèmes, il est aisé de trouver la valeur optimale du problème quand la quantité de données à traiter est raisonnable et l'outil de calcul a de bonnes capacités. Notre résolution exacte atteint ses limites avec un certain nombre de centres de données et un certain nombre de fonctions. Pour palier à cela, nous avons voulu faire un compromis entre le temps et la qualité de solution. Nous avons dans un premier temps approché le problème à l'aide d'une heuristique gloutonne, ensuite nous avons décidé d'améliorer le résultat de celle-ci avec une approche algorithmique Tabou. Dans la suite du chapitre, nous utiliserons la première partie pour décrire en détail comment fonctionnent chacune des heuristiques. Nous présenterons ensuite les résultats obtenus avec celles-ci en mettant un point particulier sur la comparaison entre elles et avec la solution exacte. Enfin nous conclurons le chapitre.

5.2 Algorithmes

Dans cette section, nous expliquerons le fonctionnement global du programme de résolution avec les heuristiques du problème, ensuite nous entrerons en détail pour décrire les deux heuristiques.

5.2.1 Algorithme général de résolution du problème

La figure 5.1 montre comment les heuristiques sont combinées pour donner une solution à chaque heure. L'algorithme est celui du placement d'une seule chaîne dans le système. Nous plaçons chaque chaîne exactement comme dans la résolution avec AMPL. Nous retenons la solution de chaque placement pour placer les autres chaînes plus tard.

La première étape est la construction des listes de centre de données qui peuvent héberger chaque fonction de la chaîne en cours d'assignation. En effet, il y a plusieurs fonctions possibles mais les chaînes ne sont en réalité constituées que de certaines d'entre elles. Il ne sert donc à rien de travailler avec les centres qui ne peuvent contenir aucune des fonctions de la chaîne.

Cette opération permet de diminuer le nombre de centres de données sur lesquels on va faire les permutations et les classements dans les algorithmes suivants.

Dans la deuxième étape, on initialise l'heure h. En effet puisque les chaînes entrent et sortent

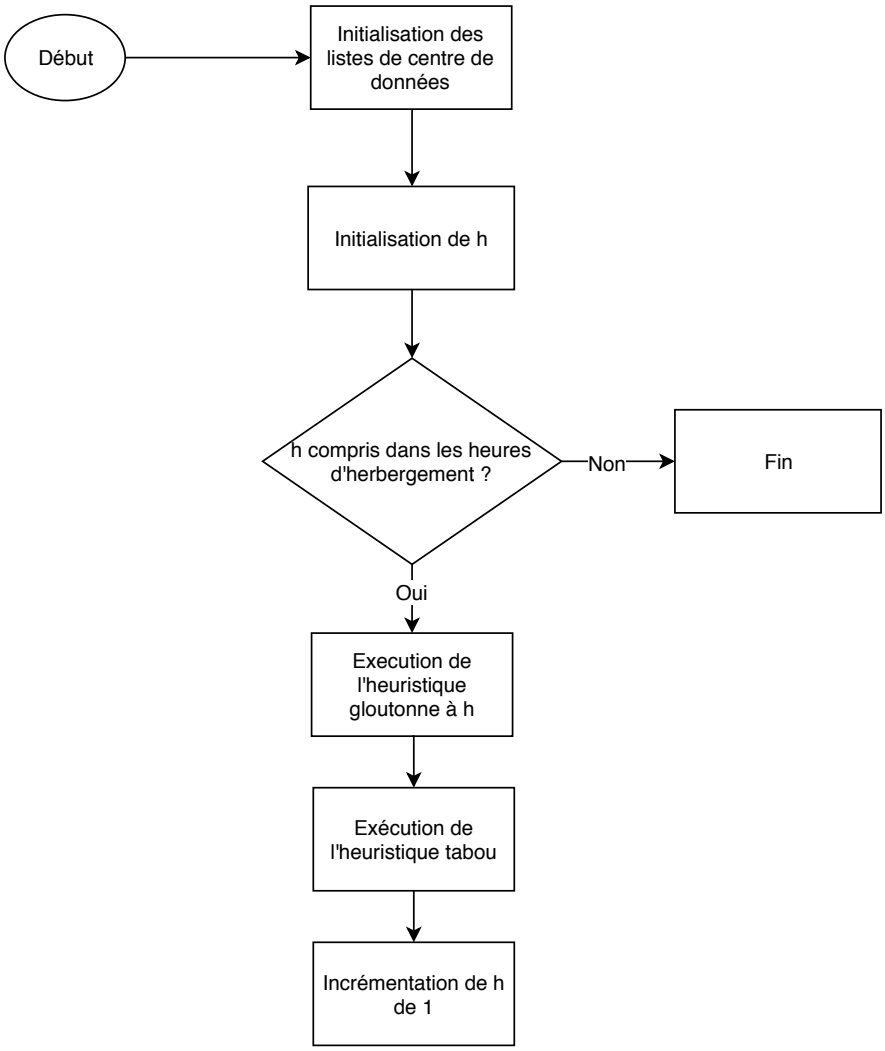


Figure 5.1 Algorithme Générale de résolution du problème

à un moment donné, il faut vérifier qu'on initialise bien l'heure de début à l'heure où la chaîne est placée pour la première fois. On doit après cela mettre une boucle et vérifier si l'heure h n'est pas plus grande que la dernière heure à laquelle la chaîne doit être assignée.

Dans la troisième étape, on exécute l'algorithme glouton pour avoir un résultat réalisable et qui respecte les contraintes du client et du problème. En fonction des placements précédents, cet algorithme soit peut trouver un résultat soit rejeter la chaîne si les contraintes sont irréalisables. Si le chaîne est rejetée, alors le système place la chaîne suivante.

Dans la quatrième étape, nous nous servons des résultats de la précédente étape pour pouvoir améliorer la solution obtenue à l'aide d'un algorithme Tabou. Nous décrirons ce Tabou dans la suite.

5.2.2 Approche gloutonne

Dans la plupart des résolutions de problème avec l'approche Tabou, un effort est mis pour avoir une bonne solution de départ. Notre objectif était donc d'avoir la meilleure solution de départ possible. Nous avons pour cela opté pour une solution itérative qui placerait chaque fonction une après l'autre. La figure 5.2 montre les différentes étapes de résolution du problème de manière itérative.

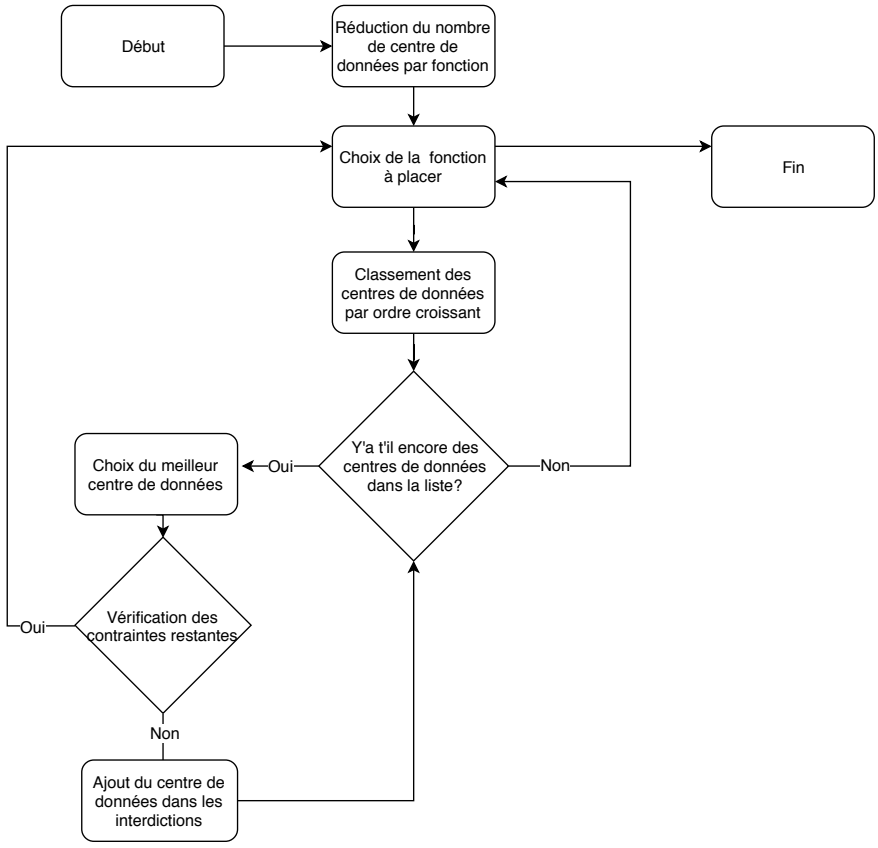


Figure 5.2 Approche gloutonne

Étape 1 : réduction du nombre de centre de données

La première étape de cette approche consiste à réduire d'avantage le nombre de centre de données sur lesquels on doit travailler. Moins on a de centre de données possibles, plus rapide est la résolution du problème. La première réduction que nous avons faite ne dépendait pas de l'heure. Elle dépendait uniquement de la capacité d'un centre de données à héberger une fonction. La deuxième réduction est dépendante de l'heure car elle tient compte des contraintes telles la migration, la symétrie, l'incompatibilité et l'affinité. Ces restrictions

comme nous l'avons expliqué au chapitre précédent sont dépendantes de l'heure à laquelle on place la fonction.

Réduction par symétrie : la réduction par symétrie consiste à vérifier qu'une chaîne symétrique à celle que nous voulons assigner est déjà assignée. Lorsqu'une chaîne est symétrique à une chaîne qui a déjà été placée, la fonction concernée par la symétrie aura comme seule ensemble de centre de données solution le centre de données auquel est assignée sa fonction jumelle dans l'autre chaîne.

Réduction par incompatibilité : la réduction par incompatibilité permet de retirer sur toutes les chaînes qui sont déjà placées les positions des fonctions incompatibles. Ainsi, ces centre de données sont retirés dans les listes des fonctions concernées dans la chaîne que le système est en train d'assigner.

Réduction par affinité : lorsque deux fonctions sont affines, elles doivent être dans le même centre de données. Pour cela, une étape qui permet d'accélérer le choix du centre de données est de créer une liste commune pour ces fonctions. Cette liste est obtenue par l'intersection des ensembles de centre de données possibles pour les deux fonctions.

Réduction par migration : une fonction à une heure précise peut avoir une restriction de migration. Pour cela, pour une heure h, si la chaîne a été placé à l'heure $h-1$, l'ensemble des centres de données possibles pour placer cette fonction est réduite au centre de données de l'heure précédente.

Étape 2 : choix de la fonction à placer

Le choix de la fonction à placer qui est montré à figure 5.2 dépend de plusieurs facteurs.

Choix croissant Il consiste à choisir les fonctions dans l'ordre où elles apparaissent dans la chaîne. Lorsqu'une fonction est placée, le système choisit la fonction qui suit dans la chaîne pour la placer. Lorsqu'une fonction est choisie dans ce cas, la liste des centres de données interdits est réinitialisée.

Choix décroissant Lorsque, pour une raison quelconque, on ne parvient pas à placer la fonction en cours et que l'ensemble des centres de données possibles est épuisé, on va aller choisir la fonction précédente dans la chaîne et essayer de la replacer. Pour cela, on va ajouter

à la liste des centres de données interdits pour la fonction le centre de données sur lequel la fonction était précédemment placée.

Lorsqu'il y n'a plus de centre de données à placer, l'algorithme s'arrête.

Étape 3 : classement par ordre croissant

L'expérience et l'intuition indiquent que moins on utilise de liens moins le coût de placement est élevé. Pour pouvoir exploiter cela, nous avons décidé de mettre le plus de fonctions possibles dans un même centre de données. Puisque le classement se fait par fonction, nous décidons de classer par ordre croissant les centres de données qui peuvent prendre la plus longue suite de fonctions possibles, au coût le plus bas possible en commençant par la fonction en cours de placement.

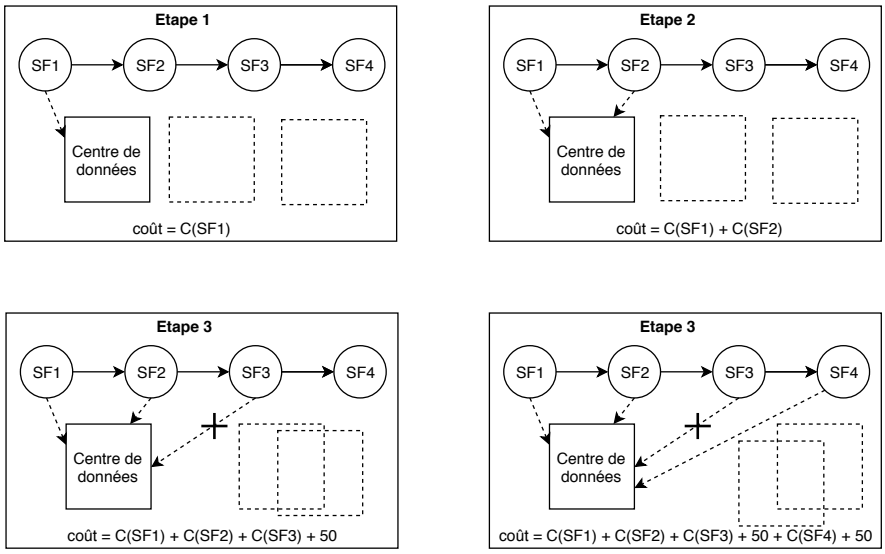


Figure 5.3 Explication du choix croissant

Par exemple, supposons que nous voulons placer une chaîne de quatre fonctions SF1, SF2, SF3 et SF4 dans cet ordre. Cette chaîne doit être placée dans un réseau de centres de données. La figure 5.3 illustre comment cela se passe.

La première fonction que l'on placera c'est la fonction SF1. Alors nous allons pour chaque centre de données vérifier s'il peut contenir SF1 si oui alors on ajoute son coût d'hébergement de SF1. On vérifie ensuite s'il peut contenir SF2 si oui, on ajoute au coût précédent le coût de SF2. On vérifie ensuite s'il peut contenir SF3. Supposons que le centre de données précédent ne peut pas, alors la suite va se briser et nous allons pénaliser cette incapacité en ajoutant au coût précédent une valeur constante que nous avons expérimentalement fixé à 50. Puisque la

fonction SF3 ne peut être placée sur ce centre de données, la fonction SF4 elle aussi va subir la même pénalisation même si ce centre de données peut la contenir, parce que la suite est brisée au niveau de la fonction 3. Les centre de données sont ensuite classés en fonction de ces coûts par ordre croissant.

Lorsque la fonction à placer n'est pas la première fonction de la chaîne, le même raisonnement que précédemment s'applique sauf que cette fois dans le coût on ajoute aussi le coût du lien qu'il faut pour quitter le centre données de la fonction précédant celle qui est entrain d'être placé vers le centre de données où on calcule le coût.

Dans cette étape, ce classement a uniquement lieu lorsque la liste ordonnée des centre de données pour la fonction que l'on veut placer n'existe pas. Lorsqu'elle existe, on passe juste à l'étape 4.

Étape 4 : vérification et choix du centre de données

Lorsque nous avons la liste triée des centres de données, nous retirons de cette liste les centres de données interdits. S'il y a encore des éléments dans cette liste, nous choisissons le premier centre de données. Nous vérifions si celui-ci respecte les autres contraintes que nous n'avons pas encore testées. Si oui, nous passons à la fonction suivante. Sinon, nous interdisons ce centre de données et nous passons au centre de données suivant dans la liste. Lorsque la liste est vide, nous retournons au choix des fonctions pour sélectionner la fonction précédemment classée. Par la même occasion la liste classée des centres de données de la fonction donc le classement vient d'échouer est supprimée et la liste des centres de données interdits l'est aussi.

Ce processus se répète jusqu'à l'obtention d'un résultat. Si à la fin de celui-ci il y n'y a pas de résultats possibles, la chaîne sera marquée comme rejetée.

Cet algorithme permet d'avoir un résultat approximativement bon dans les cas les plus compliqués que le modèle mathématique ne peut pas résoudre. Mais la solution que nous obtenons est loin de la solution optimale. Pour obtenir une meilleure solution, nous couplons cette approche avec une résolution par tabou.

Algorithm 1 Recherche Tabou pour le placement des chaînes

function TabuSearch(N_v^k, N, SolutionInitiale)
$S \leftarrow SolutionInitiale$
$Best \leftarrow S$
$L \leftarrow \{\}$
$j \leftarrow 0$
repeat
Choisir la paire $(i \in N_v^k, n \in N)$ qui ne se trouve pas dans L et qui minimise l'objectif z dans S et qui respecte les contraintes.
Ajouter $(i, S[i])$ dans L pendant $\frac{\sqrt{|V(S)|}}{2}$ itérations.
$S[i] \leftarrow n$
$j \leftarrow j + 1$
if $z(S) < z(Best)$ **then**
$Best \leftarrow S$
$j \leftarrow 0$
until $j == MAX\_ITERATIONS$
return $Best$

5.2.3 Approche Tabou

Algorithme, espace de solution et gestion des contraintes

La solution initiale que nous envoyons a l'algorithme Tabou est un tableau qui contient comme clé les fonctions et comme valeurs les centres de données vers lesquels ces fonctions sont assignées, une chaîne à la fois.

Cette solution respecte toutes les contraintes présentées dans le modèle au chapitre 3. L'objectif z sur lequel on se base pour faire les comparaisons des mouvements est celui du modèle au chapitre 3.

Pour pouvoir gérer les contraintes de symétrie, d'affinité, d'incompatibilité et celle sur les fonctions assignées une fois arrivé au Tabou, on réduit pour chaque fonction la liste des centres de données possibles quand ceux-ci ne peuvent pas contenir la fonction.

Pour la contrainte de délai, on vérifie à chaque itération ce qui s'ajoute ou se retire sur le délai global à chaque mouvement.

Pour la contrainte de capacité des centres données, la capacité résiduelle des centres données est calculée à chaque placement des chaînes et chaque fois qu'une fonction peut être assignée à un centre de données, on vérifie si cette fonction ne va pas dépasser la capacité du centre de données.

Relation de voisinage

A chaque itération, un centre de données se déplace d'une solution à une autre qui est sa voisine. Une solution S' est voisine d'une autre solution S si et seulement si elles diffèrent sur le placement d'une fonction. Une exception est faite pour les fonctions affines d'une chaîne. Si deux fonctions d'une chaîne sont affines, une solution qui déplacerait ces deux fonctions vers le même centre de données est considérée comme une solution voisine dans notre recherche Tabou. Le voisinage de notre solution est appelé $V(S)$.

Dans notre Tabou, le mouvement que nous effectuons est celui qui minimise le plus la fonction objectif et qui respecte les contraintes du problème. Pour pouvoir comparer la qualité d'une solution par rapport à une autre, nous conservons en mémoire le coût qui s'ajouterait si on retirait une fonction de son centre de données dans S. Ensuite à chaque centre de données visité, on calcule le coût que cette fonction ajoute à la fonction objectif. Nous conservons la meilleure différence entre le coût qui est retiré et le coût qui est ajouté. Cette meilleure différence nous donne le mouvement qui minimise le plus notre fonction objectif.

Parce que la vérification des solutions se fait assez vite et que le voisinage est déjà réduit avec les contraintes du problème, nous explorons tous les voisins pour voir les meilleurs d'entre eux.

La liste Tabou

Pour éviter de tomber et de demeurer dans des optimums locaux, nous mettons en place une liste dans laquelle est conservée les déplacements déjà faits. Lorsqu'une solution voisine $S\prime$ est trouvée, la solution S devient alors cette solution $S\prime$. Le couple formé de la fonction qui a permis de passer de la solution S à la solution $S\prime$ et son centre de données dans la solution S est ajouté dans la liste Tabou (L). Cette solution est gardée dans L pendant $\frac{\sqrt{|V(S)|}}{2}$ itérations.

Le critère d'arrêt

L'heuristique s'arrête lorsque la meilleure solution n'a pas été améliorée durant MAX_ITERATION itérations. La valeur de ce nombre d'itération a été trouvée expérimentalement et équivaut au double de la somme du nombre de centre de données et de nœuds dans la chaîne qu'on place : $2(|N_v^k| + |N|)$

5.3 Résultats et comparaisons

Les données que nous avons utilisées pour faire nos tests sont générées de la même façon qu'au chapitre 4.

5.3.1 Ensembles d'expériences des petits cas et comparaisons avec la solution exacte

Nous avons testé les algorithmes heuristiques et tabou et le modèle AMPL-Cplex dans trente petits cas. A partir de ces tests, nous avons évalué le temps d'exécution et le différence relative des solutions heuristiques par rapport à la solution du modèle AMPL-Cplex. Dans ces cas, nous avons fait varier le nombre de fonctions, le nombre de centres de données et le pourcentage de fonctions qu'un centre de données peut avoir. Les différences relatives sont obtenues par les expressions suivantes :

— $DG = \frac{RG-RA}{RA} \times 100$
— $DT = \frac{RT-RA}{RA} \times 100$

Les tableaux 5.1 et 5.2 nous présentent les résultats de cette expérience.

Dans cette expérience, nous exécutons chaque cas trois fois et nous retenons le cas le moins bon rencontré.

Nous remarquons dans plus de 75% des cas étudiés que les gaps sont inférieurs à 5.5%. Dans le cas où les centres de données possèdent 50% des fonctions possibles, il est plus difficile de se rapprocher de la solution optimal et plus le nombre de centres de données augmente moins notre tabou est fiable. Par contre, dans le cas où les centres de données ont 90% des fonctions possibles, on a de meilleurs résultats. Il n'y a pas de valeur à plus de 10 % et la plupart des valeurs ont un gap de moins de 5%.

En ce qui concerne l'algorithme glouton, Nous remarquons qu'il donne dans la plupart des cas des solutions qui ont des gaps supérieurs à 50%. Dans les autres cas l'heuristique peut tomber sur la solution optimal ou bien assez proche de celle-ci. Nous remarquons de cela que l'algorithme tabou améliore de manière considérable l'heuristique gloutonne.

Légende pour les tableaux 5.1 et 5.2

— CD - Centre de données
— NF - Nombre de fonctions
— % CDF - Pourcentage des fonctions possibles dans les centres de données.
— RA - Résultats AMPL
— RG - Résultats de l'algorithme gloutonne

Tableau 5.1 Ensemble des tests ainsi que les temps moyens de leur résolution avec un CDF de 50 %

CD	NF	RA	RG	RT	GA (%)	DG (%)	DT (%)
5	3	47.8	47.8	47.8	0	0	0
5	5	2248.1	2390.8	2251.0	0	6.3	0.1
5	8	2525.8	2650.1	2631.5	0	4.9	4.1
8	3	51.7	51.7	51.7	0	0	0
8	5	1050.0	1405.2	1078.3	0	33.8	2.7
8	8	1396.0	1659.5	1427.2	0.27	18.8	2.2
10	3	618.9	782.7	627.8	0	26.4	1.4
10	5	81.8	81.8	81.8	0	0	0
10	8	2039.1	3045.0	2202.2	1.13^{-13}	49.3	7.9
15	3	36.9	47.6	37.4	0.18	28.9	1.3
15	5	800.1	907.8	884.2	0	13.4	10.5
15	8	1236.1	2186.9	1359.2	0.09	76.9	9.9
20	3	462.2	635.6	474.4	0	37.5	2.6
20	5	504.3	943.8	547.4	0.02	87.2	8.5
20	8	878.9	1820.8	995.9	0	107.2	13.3

— RT - Résultats du tabou
— GA - Gap fourni par ampl en pourcentage
— DG - Différence relative de l'algorithme gloutonne
— DT - Différence relative du tabou

5.3.2 Grands réseaux

Parce que dans nos petits cas, le cas avec des centres de données possédant 90% des fonctions possibles ont de meilleurs résultats, nous choisissons l'utiliser pour tester les grands réseaux. Nous avons testé les cas ayant entre 50 et 300 centres de données. Les résultats de ces tests se trouvent dans le tableau 5.3. La première colonne de ce tableau contient les centres de données, ensuite nous avons le nombre de fonctions que les chaînes possèdent. La troisième et la quatrième colonnes montrent respectivement les résultats de l'algorithme glouton et celui de l'algorithme Tabou. La cinquième colonne montre la différence relative entre le résultat de l'algorithme glouton et l'heuristique Tabou. La formule pour calculer cette différence est : $Différence = \frac{(RG-RT)}{RT} \times 100$. La dernière colonne montre le temps de résolution en secondes.

Les cas présentés dans ce tableau sont des cas que nous n'arrivons pas à résoudre avec Cplex-Ampl. Nous remarquons premièrement que les résultats trouvés dans ce placement sont toujours du même ordre que ceux trouvés avec les petits et moyens cas. L'heuristique

Tableau 5.2 Ensemble des tests ainsi que les temps moyens de leur résolution avec un CDF de 90 %

CD	NF	RA	RG	RT	GA (%)	DG (%)	DT (%)
5	3	43.4	695.0	43.8	0	1501.3	0.9
5	5	74.6	74.6	74.6	0	0	0
5	8	119.96	119.96	119.96	0	0	0
8	3	29.9	328.4	30.6	0	999	2.3
8	5	62.8	64.1	64.1	0	2	2
8	8	115.9	126.5	119.7	0.5	9.1	3.2
10	3	30.1	47.7	31.5	0	58.3	4.6
10	5	58.4	137.3	58.5	0	135.1	0.2
10	8	111.4	111.4	111.4	0	0	0
15	3	31.0	69.3	31.6	0	123.5	1.9
15	5	59.5	180.7	59.6	0	203.7	0.2
15	8	101.3	106.9	105.2	0.27	5.5	3.8
20	3	24.25	128.0	26.0	0	427.8	7.2
20	5	52.8	55.6	55.6	0.04	5.3	5.3
20	8	103.5	962.1	108.8	0.20	829.5	5.1

gloutonne est toujours aussi inefficace avec des gaps de plus de 100% dans la plupart des cas mais l'ajout du tabou l'améliore grandement.

Les temps de traitement ne dépassent pas 30 secondes avec des réseaux allant jusqu'à 300 centres de données.

Légende pour le tableau 5.3

- CD - Centre de données
- NF - Nombre de fonctions
- RG - Résultats de l'algorithme glouton
- RT - Résultats du tabou

Cette approche Tabou est à notre connaissance la première en la matière dans le placement des fonctions de réseaux.

5.4 Conclusion

Dans ce chapitre, nous avons présenté notre approche algorithmique pour résoudre les cas qui n'ont pas été résolus avec AMPL-Cplex. Nous avons en premier lieu présenté l'algorithme générale d'exécution du problème. Ensuite nous avons présenté l'heuristique gloutonne que nous avons conçue. Nous sommes partis de la solution de cette heuristique gloutonne pour

Tableau 5.3 Ensemble des tests ainsi que les temps moyens de leur résolution avec un CDF de 90 % pour des larges cas.

CD	NF	RG	RT	Différence (%)	Temps (s)
50	3	196.9	22.6	769.3	1.8
50	5	45.7	43.5	5.1	2.9
50	8	456.9	113.4	302.9	5.6
100	3	29.1	25.2	15.4	3.9
100	5	592.42	55.4	969.3	7.1
100	8	1454.0	118.1	1131.1	10.5
200	3	50.1	22.0	127.7	8.1
200	5	77.6	60.9	27.4	12.2
200	8	153.8	153.5	0.2	17.7
300	3	266.7	27.6	866.3	11.3
300	5	266.2	51.5	416.9	18.9
300	8	543.7	107.8	404.3	25.4

construire une meilleure solution avec la recherche Tabou, que nous avons présentée. Enfin, nous avons exposé les résultats comparatifs de ces heuristiques avec ceux du modèle au chapitre 4. Le problème est résolu rapidement même dans des cas que AMPL-Cplex ne peut pas résoudre. Lorsque les centres de données peuvent contenir 50% des fonctions possibles, nous avons quelques cas où les différences relatives sont supérieurs à 5%. Mais dans les cas à 90%, il y a de bien meilleurs résultats. Ce qui nous pousse à penser que plus il y a de choix de centre de données lors des mouvements dans le Tabou meilleurs sont les résultats. Nous avons aussi testé les cas de grands réseaux ayant entre 50 et 300 centres de données. Les temps d'éxécution ne sont pas supérieurs à 30 secondes.

Ces résultats peuvent être améliorés. Pour ce faire, plusieurs pistes s'offrent. Notamment une des pistes est la diversification. Nous avons remarqué que la solution peut rester longtemps dans un optimum local et en sortir difficilement. Une approche pour résoudre cela serait de construire une solution différente à partir de la solution reçue de l'algorithme glouton pour explorer son voisinage.

CHAPITRE 6 CONCLUSION

Pour conclure, nous présentons dans ce qui suit une brève revue des travaux que nous avons effectués au cours de notre maitrise. Ensuite nous présentons les limites de notre solution.

6.1 Synthèse des travaux

Dans ce travail, nous avons étudié le problème de placement dynamique des chaînes de fonctions de services réseaux. Ces chaînes permettent, dans un contexte où les NFV et les SDN sont en pleine expansion de déployer des réseaux de plus en plus flexibles, fiables et portables. Elles permettent de diminuer les coûts et de simplifier les réseaux actuels. Ce placement soulève plusieurs défis. Tout d'abord, les utilisateurs de chaînes ont besoin des placements qui respectent les qualités de service. Deuxièmement, ils doivent aussi coûter le moins possible. De plus, les soucis de sécurité des propriétaires des chaînes doivent être pris en considération.

Dans ce travail, nous avons d'abord montré que le placement des chaînes de réseaux est étudié dans la littérature. Nous avons présenté et classifié certains travaux qui ont été fait dans ce sens là. En partant de ce qui existe, nous avons contribué à cette recherche en considérant un certain nombre d'aspects absents de la littérature. Par exemple, les centres de données ont des coûts qui changent avec le temps. Cette variation de coût fait qu'un placement efficace n'est pas fixe. De plus, la demande en débit des chaînes est variable et les délais sur les liens de transmission le sont aussi. Pour ces raisons, nous avons pensé à faire un placement dynamique des chaînes de fonctions. En plus de ce dynamisme, nous avons défini la symétrie et l'incompatibilité qu'il peut y avoir entre les chaînes. Nous avons aussi défini l'affinité entre les fonctions réseaux et leurs migrations pour tenir compte des soucis de sécurité que peuvent avoir les propriétaires des chaînes.

Pour résoudre ce problème, nous avons défini au chapitre 3 les concepts précédemment énoncés et nous avons proposé un programme linéaire en nombre entier qui a pour objectif de minimiser le coût de placement dynamique de chaque chaîne soumise au système en respectant les contraintes de délais imposés par les propriétaires des chaînes, les contraintes de capacités au niveau des centres de données, et en tenant compte des restrictions de symétrie, d'incompatibilité, d'affinité et de migration. En exécutant ce modèle avec AMPL-Cplex, nous avons remarqué qu'avec des réseaux de moins de 40 centres données et des chaînes d'environ 8 fonctions, les résultats sont obtenus dans des temps raisonnables. De plus, en faisant une

comparaison des coûts qu'apportent les restrictions, nous avons remarqué que la contrainte de symétrie ne génère presque pas de coût, la contrainte d'affinité est plus couteuse que la contrainte de migration, et que la contrainte d'incompatibilité entre les chaînes est la plus couteuse de toutes. Enfin, cette même contrainte d'incompatibilité est à l'origine du plus grand nombre de rejet de chaînes.

Pour résoudre le problème pour de grands réseaux, nous avons conçu deux heuristiques. Une heuristique gloutonne qui a pour but de donner un résultat réalisable rapidement. Le résultat de cette heuristique est passé ensuite à l'heuristique Tabou que nous avons proposée. Les solutions de ces heuristiques et celles obtenues par le modèle linéaire ont été comparées. En particulier en considérant deux types de centres : les centres "spécialisées" qui contiennent un nombre restreints de fonctions réseaux et ceux non spécialisés. Ces comparaisons nous ont montré que moins il y a de centre de données spécialisées plus la résolution Tabou est efficace. Nous avons notamment eu des différences relatives inférieur à 5.5% quand les centres de données peuvent héberger 90% des types de fonctions possibles présent dans nos tests.

6.2 Limites de la solution proposée

Les principales limites de ce travail se situent au niveau de l'obtention de résultats proche de l'optimalité lorqu'on a des centres de données plus spécialisés. En effet, le gap de résultat est plus grand lorsque les centres de données vont avoir environ 50% des fonctions proposées. Une autre limite est le fait que nous faisons une hypothèse que les chaînes sont ordonnées dès le départ, alors l'ordonnancement lui même pourrait faire l'objet d'optimisation. Enfin l'hypothèse selon laquelle le flux est conservé peut ne pas être respectée dans des cas où une fonction de service va compresser les paquets ou les bourrer.

RÉFÉRENCES

B. Addis, D. Belabed, M. Bouet, et S. Secci, “Virtual network functions placement and routing optimization”, dans *Proc. IEEE 4th International conference on Cloud Networking*. IEEE, Oct. 2015, pp. 171–177. DOI : 10.1109/cloudnet.2015.7335301

Z. Allybokus, N. Perrot, J. Leguay, L. Maggi, et E. Gourdin, “Virtual function placement for service chaining with partial orders and anti-affinity rules”, *Networks*, vol. 71, no. 2, pp. 97–106, Sep. 2017. DOI : 10.1002/net.21768

L. Askari, A. Hmaity, F. Musumeci, et M. Tornatore, “Virtual network function placement for dynamic service chaining in metro-area networks”, dans *Optical Network Design and Modelling*. IEEE, Mai 2018, pp. 136–141. DOI : 10.23919/ondm.2018.8396120

M. F. Bari, S. R. Chowdhury, R. Ahmed, et R. Boutaba, “On orchestrating virtual network functions”, dans *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, Nov. 2015, pp. 50–56. DOI : 10.1109/cnsm.2015.7367338

M. Beck, J. Botero *et al.*, “Scalable and coordinated allocation of service function chains”, *Computer communication*, vol. 102, pp. 78–88, Avr. 2017.

A. Bednarz, “Top reasons for network downtime”, 2016. En ligne : https://www.networkworld.com/article/3142838/infrastructure/top-reasons-for-network-downtime.html

D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, et H. A. Chan, “Optimal virtual network function placement in multi-cloud service function chaining architecture”, *Computer communications*, vol. 102, pp. 1–16, Fév. 2017.

M. Bouet, J. Leguay, T. Combe, et V. Conan, “Cost-based placement of vDPI function in NFV infrastructures”, *International Journal of Network Management*, vol. 25, pp. 490 – 506, Nov. 2015.

P. Chowdhury, B. Mukherjee, S. Sarkar, G. Klamer, et S. Dixit, “Hybrid wireless-optical broadband access network (WOBAN) : prototype development and research challenges”, *IEEE Network*, vol. 23, no. 3, pp. 41–48, Mai 2009.

A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, et X. Hesselbach, “Virtual network embedding : A survey”, *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp.

1888–1906, 2013. DOI : 10.1109/surv.2013.013013.00155

J. M. Halpern, C. Pignataro *et al.*, “Service Function Chaining (SFC) Architecture”, RFC 7665, Oct. 2015. DOI : 10.17487/rfc7665. En ligne : https://rfc-editor.org/rfc/rfc7665.txt

Journal du Net, “Nombre d’internautes dans le monde”, 2018. En ligne : https://www.journaldunet.com/ebusiness/le-net/1071539-nombre-d-internautes-dans-le-monde/

Juniper Networks inc., “Energy efficiency for network equipment : Two steps beyond greenwashing”, Août 2008. En ligne : http://www.juniper.net/solutions/literature/white_papers/200284.pdf

H. Ko, D. Suh, H. Baek, et S. Pack, “Optimal placement of service fucntion in service function chaining”, dans *Proc. IEEE International Conference on Ubiquitous and Future Networks*, Juil. 2016, pp. 102–105.

X. Lin, C. Qian *et al.*, “The virtual network function placement problem”, dans *Proc. IEEE International Conference on computer communications*, Avr. 2015.

M. C. Luizelli, L. R. Bays, et L. S. Buriol, “Piercing together the nfv provisioning puzzle : Efficient placement and chaining of virtual network functions”, dans *IEEE International Symposium on integrated Network Management*. IEEE, Mai 2015, pp. 98–106. DOI : 10.1109/inm.2015.7140281

M. Mechtri, C. Ghribi, et D. Zeghlache, “A scalable algorithm for the placement of service function chains”, *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, sep 2016. DOI : 10.1109/tnsm.2016.2598068

S. Mehraghdam, M. Keller, et H. Karl, “Specifying and placing chains of virtual network functions”, dans *Proc. IEEE International conference on Cloud Networking*, Oct. 2014, pp. 7–13.

P. Morreale, J. Anderson *et al.*, *Software Defined Networking Design and Deployment*. CRC Press, 2014.

T. Nadeau, P. Quinn *et al.*, “Problem Statement for Service Function Chaining”, RFC 7498, Avr. 2015. DOI : 10.17487/rfc7498. En ligne : https://rfc-editor.org/rfc/rfc7498.txt

N. Tastevin, M. Obadia, et M. Bouet, “A graph approach to placement of service functions chains”, dans *IFIP/IEEE Symposium on Integrated Network and Service Management*. IEEE, Mai 2017, pp. 134–141. DOI : 10.23919/inm.2017.7987273

L. Wang, Z. Lu, X. Wen, R. Knopp, et R. Gupta, “Joint optimization of service function chaining and resource allocation in network function virtualization”, *IEEE Access*, vol. 4, pp. 8084–8094, Nov. 2016.

W. Xia, Y. Wen, C. H. Foh, D. Niyato, et H. Xie, “A Survey on Software-Defined Networking”, *IEEE Communication Surveys and Tutorials*, vol. 17, pp. 27–51, 2017.

B. Yi, X. Wang, K. Li, S. k. Das, et M. Huang, “A comprehensive survey of network function virtualization”, *Computer Networks*, vol. 133, pp. 212–262, Mars 2018. DOI : 10.1016/j.comnet.2018.01.021